An Abstract Automata Learning Framework

Gerco van Heerdt

June 28, 2016

Master Thesis
Computing Science
Radboud University Nijmegen

First Supervisor
Prof. Dr. F.W. Vaandrager
f.vaandrager@cs.ru.nl

Second Supervisor
MSc J.S. Moerman
joshua.moerman@cs.ru.nl

External Supervisor
Dr. A. Silva
alexandra.silva@ucl.ac.uk

Second Assessor
Prof. Dr. B.P.F. Jacobs
bart@cs.ru.nl

Abstract

Advanced applications of automata learning demand increasinly complex learning algorithms that are hard to reason about. We use the language of category theory to develop a unifying framework for the study of automata learning and show that by a slight generalization minimization and conformance testing are covered as well. The results are expected to inspire or even form the basis of new algorithms. As an example, we instantiate the framework to set the first steps towards an algorithm that learns nominal automata.

Contents

1 Introduction

To help increase confidence in the correctness of a computer system, many methods have emerged, some of which use a simplified automaton model of the system. For example, in *model checking* [31, 16] one verifies properties for the model, which is relatively efficient compared to directly testing the system. In fact, there may be no analogous testing method available at all to determine whether a property holds for the system. Unfortunately, constructing a model manually often requires a lot of time from a person having expert knowledge of the system.

Active automata learning is a technique that infers an automaton of a system just by providing inputs and observing the corresponding outputs, thus enabling the use of model checking on arbitrary systems [58]. Moreover, the ability to extract models has found many applications in analyzing the differences of systems that are supposed to perform the same task.

The classical automata learning algorithm due to Angluin [7] reduces the learning problem to the problem of determining equivalence of an automaton with the system. That is, the learner repeatedly constructs a *hypothesis* automaton and asks an oracle if it is correct. If not, a *counterexample*, i.e., an input that witnesses the inequivalence of the hypothesis with the system, is provided, which is used by the learner to refine the hypothesis. In practice, determining the equivalence of a hypothesis with the system has to be approximated using a finite number of tests. The implementation of this algorithm and its derivatives in tools such as LearnLib [45] opened the doors to numerous applications.

An application that has been explored recently is the rejuvenation of legacy components. Schuts et al. [63] learned automata of both the new and the old implementation of software that in a real development project at Philips had to be replaced as a result of the introduction of new hardware. Using these learned models, they were able to identify a discrepancy that had not been found using the existing unit test cases. This discrepancy corresponded to an issue in the new implementation. Interestingly, they also found a mistake in the old implementation. By adjusting the implementations, the process could be repeated, and eventually the implementations were equal (up to the approximations used by the learning algorithm).

One of the major uses of automata learning that has been getting attention for over a decade now is the comparison of implementations of a network protocol with one another and with their specification. For example, de Ruiter and Poll [32] found new flaws in three out of nine TLS implementations for which they manually inspected learned models. Cho et al. [29] gained new insights into botnet protocols by inferring models for them. They also observed that learning may be used for fingerprinting by extracting from a learned model behavior that is unique to the corresponding implementation. This same observation was made by de Ruiter and Poll: all of the evaluated TLS implementations exhibited unique behavior.

When the target is a network protocol, it is relatively easy for a computer program to send inputs and observe outputs. Chalupar et al. [27] show that there are also applications to systems that require physical interaction: they built a Lego robot to operate smartcard readers used for Internet banking, and using this setup inferred automata for these devices. With this approach they confirmed a previously known vulnerability in one of the readers.

A final important observation is that whereas testing methods usually yield at most one counterexample, the model obtained through automata learning allows to extract a description of *all* invalid behavior. This observation was put to practice by Chapman et al. [28], who learn a description of the errors in a program up to a user-defined abstraction.

Automata learning is well understood in its classical setting of *deterministic automata* and trivial generalizations thereof. However, the network protocols mentioned earlier make use of entities such as sequence numbers that in this formalism would result in an infinite number of states. Aarts et al. [1] advocate the use of a *mapper* that abstracts from these entities so that a finite model can still be learned. Constructing such a mapper manually requires time and expert knowledge of the protocol. Subsequent research has therefore gone into automating this process [38, 2].

The expressiveness of the automata learned through these automatically refined abstractions was still more limited than desired. Although progress has been made in this respect [3], much of the further research has been directed towards adapting the learning algorithm to directly learn so-called *register automata* [25, 40, 39, 23, 26], where states are equipped with registers that on a transition are assigned and can be compared with the symbol that is being read using a suitable predefined set of operations. Unfortunately, these algorithms have become rather complicated. Moreover, although the algorithms seem to use properties adapted from Angluin [7], which is also true of algorithms learning other types of automata [e.g. 22, 8], no unifying formalism has been developed to study automata learning independent of the targeted automaton type. Without an abstract framework that captures results irrelevant of the type of automata involved, there is little guidance to the development of algorithms in yet unexplored settings. Firm abstract results could be of great help in designing the increasingly complicated learning algorithms.

In this thesis, we lay the foundations of an automata learning framework using the language of *category theory* by expanding on ideas first explored by Jacobs and Silva [46]. Category theory is capable of both abstracting from and generalizing mathematical structures. In computer science, many concepts can be described as *algebras* or *coalgebras*, both of which have been studied thoroughly in a categorical setting [4, 62]. The fields of algebra and coalgebra come together in the study of automata.

The use of an abstract language does imply that no detailed general algorithms can be developed. Our focus is on the core of the learning algorithm. We provide a fully abstract characterization of the data structures used in learning and study on this level the properties that allow for the construction of a hypothesis. Moreover, we identify sufficient conditions for the hypothesis to be correct. These results do not directly induce a generalization of Angluin's algorithm. In particular, they do not suggest what should be done with a counterexample for a hypothesis. However, we show that some things can still be said on this topic and that in the specific cases studied the full algorithm follows without too much effort.

Although we use the language of category theory to derive results with maximum applicability, no advanced category theoretical results will be applied. Section 2 introduces all the necessary concepts and proceeds to develop a basic theory of automata on an abstract level. This is merely a simplified presentation of the pioneering work by Arbib and Manes [10, 11]. An important topic is the theoretical construction of the unique minimal automaton accepting a given language.

Based on this theory of automata, we develop an abstract framework for the conceptual study of automata learning in Section 3. We show in detail how the main data structures used in active learning of deterministic automata fit in this framework. In Section 4 we observe that, by a simple generalization, our framework can also be used to study minimization procedures. Although the automata learning algorithm by Angluin is known to approximate the minimal automaton for the target language, no explicit formalism that generalizes the data structures used in learning and minimization seems to have been developed before. After explaining how reachability analysis and state merging procedures formally relate to concepts in automata learning, we shortly discuss in Section 4.1 how a specific kind of *conformance testing* fits in the developed framework as well.

It is often observed that Angluin's algorithm can straightforwardly be generalized to an arbitrary output set. We briefly discuss this generalization in Section 5, where we also review some practical optimizations.

More category theory is introduced in Section 6, where we lift our basic assumptions to the category of algebras over a *monad*. Finally, Section 7 instantiates these results to the category of sets equipped with a group action for a given group, which we study in more detail. This setting is made non-trivial by allowing certain infinite groups. The result is a first automata learning algorithm for languages accepted by *nominal automata* [21]. Thus, we provide a starting point for a generalization of register automata based in a theory that is being studied extensively.

Specific related and future work will be discussed in Sections 3.4, 6.3, and 7.4, and some final considerations are given in Section 8.

2 Automata Theory, Categorically

Category theory provides an abstract framework in which many mathematical structures can be studied uniformly. In computer science and engineering, automata are fundamental structures that formalize a vast range of computational models. A desire to study these different models simultaneously naturally motivated research into rephrasing automata theoretical results in the language of category theory. An inclination towards this can already be found in a paper by Arbib and Zeiger [14], and subsequently Arbib and Manes set forth a series of articles that lifted results to a much more abstract level [10, 11, 12, 13]. Arguably their most successful target was a characterization of minimality and the existence of unique minimal automata, which recently led to a generalization of a peculiar minimization algorithm that turns out to exploit a duality that had been investigated by the aforementioned authors [24].

In this section we tailor the theory of Arbib and Manes to our subsequent purposes. We assume familiarity with basic set theory and regular languages, but no knowledge about category theory is required. Therefore, we recall in Section 2.1 the basic definitions and facts needed to make this document self-contained. For a comprehensive exposition the reader is referred to the excellent works by Mac Lane [51] and more recently Awodey [15]. Section 2.2 describes automata abstractly, and the concept of a language is investigated in Section 2.3 to prepare for the identification of minimal automata in Section 2.4.

2.1 Basics

Definition 2.1 (Category)**.** A *category* $\mathbf{C}$ comprises a class $\mathsf{Obj}_{\mathbf{C}}$ of objects and for all objects A and B in $\mathsf{Obj}_{\mathbf{C}}$ a class $\mathsf{Hom}_{\mathbf{C}}(A, B)$ of *morphisms*, which can be composed in the following sense: if $f \in \mathsf{Hom}_{\mathbf{C}}(A, B)$ and $g \in \mathsf{Hom}_{\mathbf{C}}(B, C)$, then there exists a morphism $g \circ f \in \mathsf{Hom}_{\mathbf{C}}(A, C)$. Composition is required to be associative—$(h \circ g) \circ f = h \circ (g \circ f)$—and for each object A there must be an *identity* *morphism* $\mathsf{id}_A \in \mathsf{Hom}_{\mathbf{C}}(A, A)$, in such a way that $f \circ \mathsf{id}_A = f = \mathsf{id}_B \circ f$ for $f \in \mathsf{Hom}_{\mathbf{C}}(A, B)$.

If the intended category $\mathbf{C}$ is clear, we write $f\colon A \to B$ in favor of $f \in \mathsf{Hom}_{\mathbf{C}}(A, B)$. Alternatively, we may write $A \xrightarrow{f} B$, which in particular allows for the diagrammatic reasoning that often speaks more to the imagination than ordinary equations. For instance, we could have expressed the identity equation $f \circ \mathsf{id}_A = f = \mathsf{id}_B \circ f$ by stating that the diagram

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "\mathsf{id}"'] \arrow[dr, "f"] & B \arrow[d, "\mathsf{id}"] \\
A \arrow[r, "f"'] & B
\end{tikzcd}$$

must *commute*, which means that for all nodes X and Y in the diagram there is at most one distinct path from X to Y when these paths are interpreted as morphisms $X \to Y$ by folding sequences (in inverse order) using composition.

The category **Sets** has all sets as its objects, and for sets A and B the morphisms $\mathsf{Hom}_{\mathbf{Sets}}(A, B)$ are simply the functions with domain A and codomain B. For sets A, B, and C and functions $f\colon A \to B$ and $g\colon B \to C$, we have the usual composition

$$(g \circ f)(a) = g(f(a))$$

for each $a \in A$, and the identity morphism on a set A is the identity function $\mathsf{id}_A(a) = a$. The convention is that function application binds stronger than composition. The identity laws are seen to hold by noting that

$$\begin{aligned}
&(f \circ \mathsf{id}_A)(a) = f(\mathsf{id}_A(a)) = f(a) \text{ and} \\
&(\mathsf{id}_B \circ f)(a) = \mathsf{id}_B(f(a)) = f(a),
\end{aligned}$$

from which we conclude $f \circ \mathsf{id}_A = f = f \circ \mathsf{id}_B$, and for associativity we have

$$\begin{aligned}((h \circ g) \circ f)(a) &= (h \circ g)(f(a)) = h(g(f(a))) \\ &= h((g \circ f)(a)) = (h \circ (g \circ f))(a).\end{aligned}$$

We keep **Sets** as our main example of a category throughout much of this thesis, where all categories of importance have sets with possibly additional structure as objects and structure preserving functions, *homomorphisms*, between those as morphisms.

What can be described as a homomorphism of categories is called a *functor*. These allow us to describe relations between categories and sometimes to derive more structured categories at an abstract level.

Definition 2.2 (Functor)**.** A *functor* F from a category $\mathbf{C}$ to a category $\mathbf{D}$ assigns to each object A in $\mathsf{Obj}_{\mathbf{C}}$ an object $F(A)$ in $\mathsf{Obj}_{\mathbf{D}}$ and to each morphism $f\colon A \to B$ in $\mathbf{C}$ a morphism $F(f)\colon F(A) \to F(B)$ in $\mathbf{D}$. These must preserve identities and composition: $F(\mathsf{id}_A) = \mathsf{id}_{F(A)}$ and $F(g \circ f) = F(g) \circ F(f)$.

We denote such a functor by $F\colon \mathbf{C} \to \mathbf{D}$ and often write FA and Ff rather than $F(A)$ and $F(f)$. By convention, functor application binds stronger than composition.

If $F\colon \mathbf{C} \to \mathbf{D}$ and $G\colon \mathbf{D} \to \mathbf{E}$ are functors, then $GF\colon \mathbf{C} \to \mathbf{E}$ defined on objects by $GF(A) = G(F(A))$ and on morphisms by $GF(f) = G(F(f))$ is also a functor:

$$\begin{aligned}GF(\mathsf{id}_A) &= G(F(\mathsf{id}_A)) = G(\mathsf{id}_{FA}) = \mathsf{id}_{G(FA)} = \mathsf{id}_{GFA} \quad \text{and} \\ GF(g \circ f) &= G(F(g \circ f)) = G(Fg \circ Ff) = G(Fg) \circ G(Ff) \\ &= GFg \circ GFf.\end{aligned}$$

There is for each category $\mathbf{C}$ an *identity functor* $\mathsf{Id}_{\mathbf{C}}\colon \mathbf{C} \to \mathbf{C}$ defined by $\mathsf{Id}_{\mathbf{C}}(A) = A$ on objects and $\mathsf{Id}_{\mathbf{C}}(f) = f$ on morphisms. We simply have

$$\begin{gathered}\mathsf{Id}_{\mathbf{C}}(\mathsf{id}_A) = \mathsf{id}_A = \mathsf{id}_{\mathsf{Id}_{\mathbf{C}}(A)} \quad \text{and} \\ \mathsf{Id}_{\mathbf{C}}(g \circ f) = g \circ f = \mathsf{Id}_{\mathbf{C}}(g) \circ \mathsf{Id}_{\mathbf{C}}(f).\end{gathered}$$

For all categories $\mathbf{C}$ and $\mathbf{D}$ and a fixed object D in $\mathbf{D}$, there is a constant functor $\mathbf{C} \to \mathbf{D}$ that assigns D to all objects in $\mathbf{C}$ and id_D to each morphism in $\mathbf{C}$. The proof of functoriality is similar to the one for the identity functor, except that for preservation of compositions one uses that $\mathsf{id}_D = \mathsf{id}_D \circ \mathsf{id}_D$.

For a fixed set A, the functor $(-) \times A\colon \mathbf{Sets} \to \mathbf{Sets}$ assigns to every set B the Cartesian product $B \times A$, and given a function $f\colon B \to C$, we define $f \times A\colon B \times A \to C \times A$, written $f \times \mathsf{id}_A$ as an instance of a more general definition, by $(f \times \mathsf{id}_A)(b, a) = (f(b), a)$ for all $a \in A$ and $b \in B$. Note that indeed $\mathsf{id}_B \times \mathsf{id}_A$ is the identity on $B \times A$, and for $g\colon C \to D$,

$$\begin{aligned}((g \circ f) \times \mathsf{id}_A)(b, a) = ((g \circ f)(b), a) &= (g(f(b)), a)\\ &= (g \times \mathsf{id}_A)(f(b), a)\\ &= (g \times \mathsf{id}_A)((f \times \mathsf{id}_A)(b, a))\\ &= ((g \times \mathsf{id}_A) \circ (f \times \mathsf{id}_A))(b, a).\end{aligned}$$

Given any functor $F\colon \mathbf{C} \to \mathbf{C}$, called an *endofunctor* because its domain and codomain coincide, we can define a category $\mathbf{Alg}(F)$ of *algebras* for F. Here the objects are tuples (A, a), where A is an object and $a\colon FA \to A$ a morphism in $\mathbf{C}$. The morphisms $f\colon (A, a) \to (B, b)$ in $\mathbf{Alg}(F)$ are those morphisms $f\colon A \to B$ in $\mathbf{C}$ that make the diagram below on the left commute.

$$\begin{tikzcd}
FA \arrow[r, "Ff"] \arrow[d, "a"'] & FB \arrow[d, "b"] \\
A \arrow[r, "f"'] & B
\end{tikzcd}
\qquad
\begin{tikzcd}
FA \arrow[r, "F\mathsf{id}_A"] \arrow[d, "a"'] \arrow[dr, "a"] & FA \arrow[d, "a"] \\
A \arrow[r, "\mathsf{id}_A"'] & A
\end{tikzcd}
\qquad
\begin{tikzcd}
FA \arrow[r, "Ff"] \arrow[d, "a"'] & FB \arrow[r, "Fg"] \arrow[d, "b"] & FC \arrow[d, "c"] \\
A \arrow[r, "f"'] & B \arrow[r, "g"'] & C
\end{tikzcd}$$

We call such a morphism an F-algebra homomorphism. Recall that $F\mathsf{id}_A = \mathsf{id}_{FA}$, so that the middle diagram is easily seen to commute, which means that identities in $\mathbf{C}$ are always F-algebra homomorphisms. The composed diagram on the right shows that the composition of F-algebra homomorphisms f and g executed in $\mathbf{C}$ yields again an F-algebra homomorphism $g \circ f$:

$$g \circ f \circ a = g \circ b \circ Ff = c \circ Fg \circ Ff = c \circ F(g \circ f).$$

Together, this means that composition and identities can simply be inherited from $\mathbf{C}$.

The above calculation is easily reproduced visually in the diagram that was shown earlier on the right—a process referred to as *diagram chasing.* In proofs, we often give only a composed diagram for which individual parts are known to commute and leave the actual derivation to the reader.

There exists a *forgetful functor* $\mathbf{Alg}(F) \to \mathbf{C}$ that maps each F-algebra (A, a) to the object A in $\mathbf{C}$ and each F-algebra homomorphism $f\colon (A, a) \to (B, b)$ to the morphism $f\colon A \to B$ in $\mathbf{C}$. Preservation of identities and composition is trivial for this functor due to those in $\mathbf{Alg}(F)$ being inherited from the ones in $\mathbf{C}$.

Algebras for the functor $(-) \times A\colon \mathbf{Sets} \to \mathbf{Sets}$ are sets S equipped with a function $f\colon S \times A \to S$. One can think of this as a system with a state space given by S and that has for each state $s \in S$ and symbol $a \in A$ a successor state $f(s, a)$. The forgetful functor “forgets” about the successor information by mapping each such system to its state space.

Another endofunctor in the category of sets and functions for a fixed set A is the exponent functor $(-)^A$ that maps each set B to B^A, which is the set of functions $A \to B$; it maps each function $f\colon B \to C$ to $f^A\colon B^A \to C^A$ given by $f^A(g) = f \circ g$ for all $g\colon A \to B$. We have $\mathsf{id}_B^A(g) = \mathsf{id}_B \circ g = g$, and regarding compositions,

$$(h \circ f)^A(g) = h \circ f \circ g = h^A(f \circ g) = h^A(f^A(g)) = (h^A \circ f^A)(g).$$

Yet another endofunctor on $\mathbf{Sets}$ is the list functor $(-)^*$ that maps each set A to the free monoid A^* on A—that is, A^* contains all *words* over the *alphabet* A. Here we adopt terminology that will be useful later when talking about automata and languages. Such a word is either the empty word ε or the concatenation $u \cdot a$ of a word u with a symbol a. We extend concatenation to words and often write uv in lieu of $u \cdot v$ for words $u, v \in A^*$. The length of a word $u \in A^*$ is denoted by $|u|$. Given a function $f\colon A \to B$, the function $f^*\colon A^* \to B^*$ is defined by pointwise application of f. A proof of functoriality, which for both properties involves induction on the structure of words, is left as an exercise.

The power set functor $\mathcal{P}\colon \mathbf{Sets} \to \mathbf{Sets}$ maps each set to its power set and each function $f\colon A \to B$ to $\mathcal{P}f\colon \mathcal{P}A \to \mathcal{P}B$ given by $\mathcal{P}(f)(U) = \{f(a) \mid a \in U\}$ for each $U \subseteq A$. The functor clearly preserves identities, and for any $g\colon B \to C$ we have

$$\begin{aligned}\mathcal{P}(g \circ f)(U) &= \{(g \circ f)(a) \mid a \in U\} = \{g(b) \mid b \in \{f(a) \mid a \in U\}\} \\ &= \mathcal{P}(g)(\{f(a) \mid a \in U\}) = (\mathcal{P}(g) \circ \mathcal{P}(f))(U).\end{aligned}$$

An important notion that allows us to identify objects in a category as essentially the same is that of an *isomorphism*.

Definition 2.3 (Isomorphism)**.** A morphism $f\colon A \to B$ in some category $\mathbf{C}$ is an *isomorphism* if it has an inverse—i.e., if there exists a morphism $g\colon B \to A$ in $\mathbf{C}$ satisfying $g \circ f = \mathsf{id}_A$ and $f \circ g = \mathsf{id}_B$. In this case, the objects A and B are said to be *isomorphic* (in $\mathbf{C}$).

Note that this inverse g, sometimes denoted f^{-1}, is of course also an isomorphism. Given any object A in an arbitrary category, the identity id_A is its own inverse and thus an isomorphism: we have $\mathsf{id}_A \circ \mathsf{id}_A = \mathsf{id}_A$. In $\mathbf{Sets}$, the isomorphisms are precisely the bijective functions, and in $\mathbf{Alg}(F)$ for an endofunctor F on a category $\mathbf{C}$, the isomorphisms are precisely the F-algebra homomorphisms that are isomorphisms in $\mathbf{C}$.

The concepts of injectivity and surjectivity can also be generalized quite elegantly.

Definition 2.4 (Monos and Epis)**.** A morphism $f\colon A \to B$ in a category $\mathbf{C}$ is *monic*, or a *mono*, if for all objects X and morphisms $g_1, g_2\colon X \to A$ such that $f \circ g_1 = f \circ g_2$ we have $g_1 = g_2$; f is *epic*, or an *epi*, if for all objects X and morphisms $h_1, h_2\colon B \to X$ with $h_1 \circ f = h_2 \circ f$ we have $h_1 = h_2$.

In diagrams, we indicate monos by tailed arrows ($\rightarrowtail$) and epis by double-headed arrows ($\twoheadrightarrow$) whenever these properties are relevant.

As suggested, the monos in **Sets** are precisely the injective functions while the epis are the surjective functions.

In general, we have that an isomorphism f is monic—whenever $f \circ g_1 = f \circ g_2$ we have $g_1 = f^{-1} \circ f \circ g_1 = f^{-1} \circ f \circ g_2 = g_2$—and epic, but conversely an epic mono may not be an isomorphism. Further, if f and g are any morphisms such that $g \circ f$ is defined and epic, then g is epic: assume $h_1 \circ g = h_2 \circ g$ so that $h_1 \circ g \circ f = h_2 \circ g \circ f$, and thus because $g \circ f$ is an epi we have $h_1 = h_2$. In a similar fashion, we can show that if $g \circ f$ is a mono, then so is f.

Epis and monos may allow us to identify a unique morphism satisfying some property. For instance, suppose there exists a morphism f making the diagram below commute.

$$\begin{tikzcd}
A \arrow[r, two heads, "e"] \arrow[rrr, bend right, "g"'] & B \arrow[r, dotted, "f"] & C \arrow[r, tail, "m"] & D
\end{tikzcd}$$

A dotted arrow will always be used to represent the assertion that a morphism can be found to complete the diagram. Now if $f' \colon B \to C$ is any morphism satisfying $g = m \circ f' \circ e$, then

$$m \circ f' \circ e = g = m \circ f \circ e,$$

so because m is a mono we have $f' \circ e = f \circ e$, and using that e is epic we conclude that $f' = f$.

Uniqueness plays quite an important role in category theory, where many concepts can be brought back to the following key concept.

Definition 2.5 (Initial Object)**.** An *initial object* in a category $\mathbf{C}$ is an object A such that for any object B in $\mathbf{C}$ there exists a unique morphism $A \to B$.

An initial object itself is unique up to unique isomorphism. That is, supposing A and B are both initial in $\mathbf{C}$, there exists a unique isomorphism $A \to B$ in $\mathbf{C}$. This can be seen as follows. By initiality, there exist unique morphisms $\phi \colon A \to B$ and $\psi \colon B \to A$.

To see that these are inverse to each other, note that the identities $\mathsf{id}_A\colon A \to A$ and $\mathsf{id}_B\colon B \to B$ are unique elements of $\mathsf{Hom}_{\mathbf{C}}(A, A)$ and $\mathsf{Hom}_{\mathbf{C}}(B, B)$ respectively, which means that $\psi \circ \phi = \mathsf{id}_A$ and $\phi \circ \psi = \mathsf{id}_B$.

What could be considered the converse of this fact also works: if A is initial and B is any object isomorphic to A, then B is initial. Letting $\phi\colon A \to B$ be the isomorphism, there is for each object C a unique morphism $c\colon A \to C$, so if $f\colon B \to C$ is any morphism, then $f \circ \phi = c$ by initiality of A, and thus $f = f \circ \phi \circ \phi^{-1} = c \circ \phi^{-1}$.

In **Sets**, the initial object exists and is the empty set: there exists a unique trivial function $\emptyset \to A$ for each set A. This example may not seem very exciting, but the concept becomes powerful when additional structure is introduced. We will often describe an initial object in a category derived from a category $\mathbf{C}$ just in terms of $\mathbf{C}$, sometimes without making explicit that it concerns an initial object in some category.

Analogous to the initial object, there is also a notion of a *final object*. Instead of defining this formally, we introduce the important concept of duality.

Proposition 2.6 (Opposite Category)**.** *For each category* $\mathbf{C}$*, there is an* opposite category $\mathbf{C}^{\mathbf{op}}$ *given by*

$$\mathsf{Obj}_{\mathbf{C}^{\mathbf{op}}} = \mathsf{Obj}_{\mathbf{C}}, \qquad \mathsf{Hom}_{\mathbf{C}^{\mathbf{op}}}(A, B) = \mathsf{Hom}_{\mathbf{C}}(B, A),$$

and composition (denoted $\bullet$ *here just to contrast* $\circ$ *in* $\mathbf{C}$*) defined by* $g \bullet f = f \circ g$ *for* $f\colon A \to B$ *and* $g\colon B \to C$ *in* $\mathbf{C}^{\mathbf{op}}$*. It satisfies* $(\mathbf{C}^{\mathbf{op}})^{\mathbf{op}} = \mathbf{C}$*.*

Proof. Note that in $\mathbf{C}$ we have $f\colon B \to A$ and $g\colon C \to B$, so that $f \circ g\colon C \to A$ in $\mathbf{C}$ and thus $f \circ g\colon A \to C$ in $\mathbf{C}^{\mathbf{op}}$. This validates the definition of composition. The identity and associativity laws in $\mathbf{C}$ carry over to $\mathbf{C}^{\mathbf{op}}$:

$$f \bullet \mathsf{id}_A = \mathsf{id}_A \circ f = f = f \circ \mathsf{id}_B = \mathsf{id}_B \bullet f \quad \text{and}$$
$$(h \bullet g) \bullet f = f \circ (g \circ h) = (f \circ g) \circ h = h \bullet (g \bullet f).$$

Moreover, $(\mathbf{C}^{\mathbf{op}})^{\mathbf{op}} = \mathbf{C}$, for $\mathsf{Obj}_{(\mathbf{C}^{\mathbf{op}})^{\mathbf{op}}} = \mathsf{Obj}_{\mathbf{C}^{\mathbf{op}}} = \mathsf{Obj}_{\mathbf{C}}$ and

$$\mathsf{Hom}_{(\mathbf{C}^{\mathbf{op}})^{\mathbf{op}}}(A, B) = \mathsf{Hom}_{\mathbf{C}^{\mathbf{op}}}(B, A) = \mathsf{Hom}_{\mathbf{C}}(A, B). \qquad \square$$

A *final object* in $\mathbf{C}$ can now be defined as an object that is initial in $\mathbf{C}^{\mathbf{op}}$. Thus, A is final if and only if for each object B there exists a unique morphism $B \to A$. In **Sets** it is any singleton set. The main advantage of this definition is that we get many results for free by translating results about initial objects. For instance, final objects are unique up to isomorphism because initial objects are unique up to isomorphism, and because the isos in $\mathbf{C}^{\mathbf{op}}$ are precisely those in $\mathbf{C}$ ($g \circ f = \mathsf{id}_A$ if and only if $f \bullet g = \mathsf{id}_A$). This technique applies to many other concepts as well. For instance, an endofunctor in $\mathbf{C}^{\mathbf{op}}$ is precisely one in $\mathbf{C}$, and the monos in $\mathbf{C}^{\mathbf{op}}$ are precisely the epis in $\mathbf{C}$.

For an endofunctor $F\colon \mathbf{C} \to \mathbf{C}$ we have seen the construction of the category $\mathbf{Alg}(F)$ of F-algebras. As we have just noted, F is also an endofunctor on $\mathbf{C}^{\mathbf{op}}$, say $F^{\bullet}\colon \mathbf{C}^{\mathbf{op}} \to \mathbf{C}^{\mathbf{op}}$, and so we may construct the quite different category $\mathbf{Alg}(F^{\bullet})$, which, interpreted over $\mathbf{C}$, is the category $\mathbf{CoAlg}(F)$ of coalgebras for F. Here the objects are pairs (A, a) of an object A with a morphism $a\colon A \to FA$ in $\mathbf{C}$, and an F-coalgebra homomorphism $f\colon (A, a) \to (B, b)$ is a morphism $f\colon A \to B$ in $\mathbf{C}$ that makes the diagram below commute. Identities and composition are again inherited from $\mathbf{C}$.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "a"'] & B \arrow[d, "b"] \\
FA \arrow[r, "Ff"] & FB
\end{tikzcd}$$

There is an interesting duality exhibited by the functors $(-) \times A$ and $(-)^A$: by means of currying, we find that $\mathbf{Alg}((-) \times A)$ and $\mathbf{CoAlg}((-)^A)$ are isomorphic. Despite being another important concept in category theory, and despite being used heavily by Arbib and Manes [12], we will not cover the topic of *adjoint functors*.

2.2 Automata

Deterministic automata (DA) for an input alphabet A consist of a set of states including an initial state and a subset of accepting states, together with a function that assigns to each pair of a state and an input symbol a successor state.

In a more abstract setting, we fix a *base category* $\mathbf{B}$ and in this category an *initial state object* I, an *output object* Y, and a functor $\mathcal{D}\colon \mathbf{B} \to \mathbf{B}$ that describes the type of dynamics of our automata. We will develop the general theory abstractly while drawing motivation from the DA example.

Definition 2.7 (Automaton)**.** An *automaton* consists of a *state object* Q in $\mathbf{B}$ together with an *initial state map* $\mathsf{init}_Q\colon I \to Q$, an *output map* $\mathsf{out}_Q\colon Q \to Y$, and a *dynamics* $\delta_Q\colon Q \to \mathcal{D}Q$.

We employ here a naming scheme that avoids an overabundance of declarations. However, note that formally an automaton should be identified as a tuple $(Q, \mathsf{init}_Q, \mathsf{out}_Q, \delta_Q)$ such that the same state object may form different automata. Also note that we may leave out subscripts if the automaton is clear from the context.

A formal automaton Q is visualized in Figure 2.1 on the left. On the right, we instantiate to DA by taking $\mathbf{B}$ to be the category $\mathbf{Sets}$ of sets and functions and defining $I = 1$, $Y = 2$, and $\mathcal{D} = (-)^A$. It is well known that elements of Q correspond bijectively to functions $1 \to Q$ for a singleton set 1, say $\{*\}$, and that a subset $F \subseteq Q$ can be represented by its characteristic function $\chi_F\colon Q \to 2$, where we take $2 = \{0, 1\}$. Specifically, $\chi_F(q) = 1$ if $q \in F$ and $\chi_F(q) = 0$ if $q \notin F$. A state $q \in Q$ is thus accepting if and only if $\mathsf{out}(q) = 1$.

For instance, consider the DA depicted in Figure 2.2. We indicate the initial state by an unlabeled arrow and draw a double circle around accepting states. An arrow from state q to q' labeled by input symbol a implies that $\delta(q)(a) = q'$. The diagram of our example

$$\begin{tikzcd}
I \arrow[r, "\mathsf{init}"] & Q \arrow[r, "\mathsf{out}"] \arrow[d, "\delta"] & Y \\
& \mathcal{D}Q &
\end{tikzcd}
\qquad
\begin{tikzcd}
1 \arrow[r, "\mathsf{init}"] & Q \arrow[r, "\mathsf{out}"] \arrow[d, "\delta"] & 2 \\
& Q^A &
\end{tikzcd}$$

Figure 2.1: Formal automata

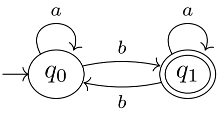


Figure 2.2: Example DA

automaton thus corresponds to the definition below.

$$Q = \{q_0, q_1\} \qquad \mathsf{out}(q_0) = 0 \qquad \delta(q_0)(a) = \delta(q_1)(b) = q_0$$
$$\mathsf{init}(*) = q_0 \qquad \mathsf{out}(q_1) = 1 \qquad \delta(q_1)(a) = \delta(q_0)(b) = q_1$$

Note that the dynamics of a DA can equivalently be described as a function $Q \times A \to Q$, which leads one to wonder if it could happen that the dynamics of some sort of automaton can only be described by a morphism $\mathcal{D}Q \to Q$. There is, however, nothing to worry about: in this case we can swap I and Y while moving to the opposite base category $\mathbf{B^{op}}$, where such a structure does fall under our definition of an automaton.

For DA one often wants to talk about the state reached after reading a word and the language accepted by each state, both of which are defined by repeatedly applying the transition function. To recover these concepts abstractly, we start dissecting the structure of our automaton model.

Definition 2.8 (Input System)**.** An *input system* is an object K in $\mathbf{B}$ equipped with morphisms $\mathsf{init}_K \colon I \to K$ and $\delta_K \colon K \to \mathcal{D}K$. A morphism $f \colon K \to L$ between input systems K and L is an *input system homomorphism* if the diagram below commutes.

$$\begin{tikzcd}
I \arrow[r, "\mathsf{init}_K"] \arrow[dr, "\mathsf{init}_L"'] & K \arrow[r, "\delta_K"] \arrow[d, "f"] & \mathcal{D}K \arrow[d, "\mathcal{D}f"] \\
& L \arrow[r, "\delta_L"] & \mathcal{D}L
\end{tikzcd}$$

Input systems and their homomorphisms form a category that inherits identities and composition from $\mathbf{B}$, very much like the categories of algebras and coalgebras that we saw in Section 2.1. A generalization of the set of all words over A now arises in this category as an initial object, which we assume to exist. That is, we assume an input system @ such that for each input system K there exists a unique input system homomorphism $r_K \colon @ \to K$ as shown below on the left.

$$\begin{tikzcd}
I \arrow[r, "\mathsf{init}_@"] \arrow[dr, "\mathsf{init}_K"'] & @ \arrow[r, "\delta_@"] \arrow[d, dotted, "r_K"] & \mathcal{D}@ \arrow[d, dotted, "\mathcal{D}r_K"] \\
& K \arrow[r, "\delta_K"] & \mathcal{D}K
\end{tikzcd}
\qquad
\begin{tikzcd}
1 \arrow[r, "\mathsf{init}_@"] \arrow[dr, "\mathsf{init}_K"'] & A^* \arrow[r, "\delta_@"] \arrow[d, dotted, "r_K"] & (A^*)^A \arrow[d, dotted, "r_K^A"] \\
& K \arrow[r, "\delta_K"] & K^A
\end{tikzcd}$$

We refer to r_K as the *reachability map* [10] of K.

In the context of DA, @ is the set A^* of finite words over A as shown in the diagram on the right. Here the initial state is the empty word, and the dynamics concatenates symbols to words:

$$\mathsf{init}_@(*) = \varepsilon \qquad\qquad \delta_@(u)(a) = ua.$$

The reachability map $r_K \colon A^* \to K$ is defined by induction on the length of words:

$$r_K(\varepsilon) = \mathsf{init}_K(*) \qquad\qquad r_K(ua) = \delta_K(r_K(u))(a).$$

For the DA in Figure 2.2 (which is an input system if we forget about out_Q) and $u \in A^*$, $r(u) = q_0$ if the number of b's in u is even; otherwise, $r(u) = q_1$.

Let us prove that r_K as defined here for the DA setting is indeed a unique input system homomorphism. To see that it is an input system homomorphism, note that

$$\begin{aligned}
(r_K \circ \mathsf{init}_@)(*) &= r_K(\varepsilon) = \mathsf{init}_K(*) \text{ and} \\
(r_K^A \circ \delta_@)(u)(a) &= r_K^A(\delta_@(u))(a) = (r_K \circ \delta_@(u))(a) = r_K(\delta_@(u)(a)) \\
&= r_K(ua) = \delta_K(r_K(u))(a) = (\delta_K \circ r_K)(u)(a).
\end{aligned}$$

For uniqueness, consider any input homomorphism $f\colon A^* \to K$. Thus, $f \circ \mathsf{init}_@ = \mathsf{init}_K$ and $f^A \circ \delta_@ = \delta_K \circ f$. We prove $f = r_K$ by induction on the length of words:

$$
\begin{aligned}
f(\varepsilon) &= f(\mathsf{init}_@(*)) = (f \circ \mathsf{init}_@)(*) = \mathsf{init}_K(*) = r_K(\varepsilon) \quad \text{and} \\
f(ua) &= f(\delta_@(u)(a)) = (f \circ \delta_@(u))(a) = f^A(\delta_@(u))(a) \\
&= (f^A \circ \delta_@)(u)(a) = (\delta_K \circ f)(u)(a) = \delta_K(f(u))(a) \\
&\overset{\text{(IH)}}{=} \delta_K(r_K(u))(a) = \delta_K(r_K(u))(a) = r_K(ua).
\end{aligned}
$$

We went from automata to input system by disregarding the output maps. If, instead, we forget about the initial state map, we arrive at *output systems.*

Definition 2.9 (Output System)**.** An *output system* is an object K in $\mathbf{B}$ equipped with morphisms $\mathsf{out}_K\colon K \to Y$ and $\delta_K\colon K \to \mathcal{D}K$. A morphism $f\colon K \to L$ between output systems K and L is an output system homomorphism if the diagram below commutes.

$$\begin{tikzcd}
& K \arrow[r, "\delta_K"] \arrow[dl, "\mathsf{out}_K"'] \arrow[d, "f"] & \mathcal{D}K \arrow[d, "\mathcal{D}f"] \\
Y & L \arrow[l, "\mathsf{out}_L"] \arrow[r, "\delta_L"] & \mathcal{D}L
\end{tikzcd}$$

We assume additionally that the category of output systems and their homomorphisms has a final object, which gives us a universe of behavior for our automata. Thus, we assume an output system Ω such that for every output system K there exists a unique output system homomorphism $o_K\colon K \to \Omega$ as shown below on the left.

$$\begin{tikzcd}
& K \arrow[r, "\delta_K"] \arrow[dl, "\mathsf{out}_K"'] \arrow[d, dashed, "o_K"] & \mathcal{D}K \arrow[d, dashed, "\mathcal{D}o_K"] \\
Y & \Omega \arrow[l, "\mathsf{out}_\Omega"] \arrow[r, "\delta_\Omega"] & \mathcal{D}\Omega
\end{tikzcd}
\qquad
\begin{tikzcd}
& K \arrow[r, "\delta_K"] \arrow[dl, "\mathsf{out}_K"'] \arrow[d, dashed, "o_K"] & K^A \arrow[d, dashed, "o_K^A"] \\
2 & 2^{A^*} \arrow[l, "\mathsf{out}_\Omega"] \arrow[r, "\delta_\Omega"] & (2^{A^*})^A
\end{tikzcd}$$

We refer to o_K as the *observability map* [10] of K.

For DA, the situation is shown on the right. In this case, Ω is the set 2^{A^*} of languages over the alphabet A,[1] equipped with an output map that determines whether a language accepts the empty word and a dynamics that computes for each input symbol the *left derivative* of the given language with respect to that symbol:

$$\mathsf{out}_\Omega(l) = l(\varepsilon) \qquad\qquad \delta_\Omega(l)(a)(v) = l(av).$$

For instance, consider for the alphabet $A = \{a, b\}$ the language $\{a, ba, aba\}$. Its left derivative with respect to a is

$$\delta_\Omega(\{a, ba, aba\})(a) = \{\varepsilon, ba\}.$$

Thus, the words that do not begin with an a disappear, and from the other words the a at the beginning is removed.

The observability map $o_K \colon K \to 2^{A^*}$ is defined by induction on the length of words: for any state $k \in K$,

$$o_K(k)(\varepsilon) = \mathsf{out}_K(k) \qquad o_K(k)(av) = o_K(\delta_K(k)(a))(v).$$

We leave the proof as an exercise.

For the DA in Figure 2.2, $o(q_0)$ is the language of all words over $\{a, b\}$ that contain an odd number of b's, whereas $o(q_1)$ is the language of all words that contain an even number of b's.

An automaton Q is both and input system and an output system, with the dynamics being shared by these substructures. We define an automaton homomorphism to be both an input and an output system homomorphism. This gives us a category **Aut** of automata and their homomorphisms such that composition and identities are inherited from **B**, as is easily checked.

We may forget about both the initial state maps *and* the output maps, in which case we are left with simple coalgebras for the functor $\mathcal{D}$. Let **Dyn** be the category of these coalgebras and their homomorphisms, which we refer to as *dynamorphisms*.

[1] In intuitive descriptions and examples we often treat languages as sets of words, whereas formal definitions work with functions $A^* \to 2$. This conversion is kept implicit.

2.3 Languages

For an automaton Q, and a DA in particular, we now have the following situation:

$$\begin{tikzcd}
I \arrow[d, "\mathsf{init}_@"'] \arrow[dr, "\mathsf{init}_Q"] & & Y \\
@ \arrow[r, dotted, "r_Q"'] \arrow[d, "\delta_@"'] & Q \arrow[ur, "\mathsf{out}_Q"] \arrow[r, dotted, "o_Q"'] \arrow[d, "\delta_Q"] & \Omega \arrow[u, "\mathsf{out}_\Omega"'] \arrow[d, "\delta_\Omega"] \\
\mathcal{D}@ \arrow[r, dotted, "\mathcal{D}r_Q"] & \mathcal{D}Q \arrow[r, dotted, "\mathcal{D}o_Q"] & \mathcal{D}\Omega
\end{tikzcd}
\qquad
\begin{tikzcd}
1 \arrow[d, "\mathsf{init}_@"'] \arrow[dr, "\mathsf{init}_Q"] & & 2 \\
A^* \arrow[r, dotted, "r_Q"'] \arrow[d, "\delta_@"'] & Q \arrow[ur, "\mathsf{out}_Q"] \arrow[r, dotted, "o_Q"'] \arrow[d, "\delta_Q"] & 2^{A^*} \arrow[u, "\mathsf{out}_\Omega"'] \arrow[d, "\delta_\Omega"] \\
(A^*)^A \arrow[r, dotted, "r_Q^A"] & Q^A \arrow[r, dotted, "o_Q^A"] & (2^{A^*})^A
\end{tikzcd}$$

We would like to talk about the semantics of the automaton as a whole—i.e., we seek an abstract characterization of the language being accepted by Q. Immediately there are two candidate definitions: we could observe the direct output after every input—$\mathsf{out}_Q \circ r_Q$—or we could observe the behavior of the initial state—$o_Q \circ \mathsf{init}_Q$. These correspond to equivalent representations of a language: a function $\mathcal{L}\colon A^* \to 2$ can also be seen as a function $\overline{\mathcal{L}}\colon 1 \to 2^{A^*}$. It turns out that this correspondence works very abstractly, and there is an important intermediate representation.

Proposition 2.10 (Language Representation)**.** *The following types of morphisms correspond bijectively to one another:*

$$\begin{array}{ll}
\mathcal{L}\colon @ \to Y & \textit{in } \mathbf{B} \\
\hline\hline
t_{\mathcal{L}}\colon @ \to \Omega & \textit{in } \mathbf{Dyn} \\
\hline\hline
\overline{\mathcal{L}}\colon I \to \Omega & \textit{in } \mathbf{B}\,.
\end{array}$$

They turn @ and Ω into automata such that $t_{\mathcal{L}}$ is an automaton homomorphism.

Proof. If we have a morphism $\mathsf{out}_@ = \mathcal{L}\colon @ \to Y$, then Ω is an automaton; if, instead, we have a morphism $\mathsf{init}_\Omega = \overline{\mathcal{L}}\colon I \to \Omega$, then @ is an automaton. We define $t_{\mathcal{L}}$ either as $o_@$ or as r_Ω in these respective cases.

$$\begin{tikzcd}
 & @ \arrow[r, "\delta_@"] \arrow[dl, "\mathsf{out}_@"'] \arrow[d, dotted, "o_@"] & \mathcal{D}@ \arrow[d, dotted, "\mathcal{D}o_@"] \\
Y & \Omega \arrow[l, "\mathsf{out}_\Omega"'] \arrow[r, "\delta_\Omega"] & \mathcal{D}\Omega
\end{tikzcd}
\qquad
\begin{tikzcd}
I \arrow[r, "\mathsf{init}_@"] \arrow[dr, "\mathsf{init}_\Omega"'] & @ \arrow[r, "\delta_@"] \arrow[d, dotted, "r_\Omega"] & \mathcal{D}@ \arrow[d, dotted, "\mathcal{D}r_\Omega"] \\
 & \Omega \arrow[r, "\delta_\Omega"] & \mathcal{D}\Omega
\end{tikzcd}$$

To avoid ambiguity, these are written as $o_{@,\mathcal{L}}$ and $r_{\Omega,\overline{\mathcal{L}}}$.

Now suppose that, instead, we have a dynamorphism $t_\mathcal{L}$. We define $\mathcal{L}$ and $\overline{\mathcal{L}}$ as the following compositions.

$$\begin{tikzcd}
I \arrow[r, "\mathsf{init}_@"] \arrow[dr, dotted, "\overline{\mathcal{L}}"'] & @ \arrow[d, "t_\mathcal{L}"] \arrow[dr, dotted, "\mathcal{L}"] & \\
 & \Omega \arrow[r, "\mathsf{out}_\Omega"'] & Y
\end{tikzcd}$$

Taking $\mathsf{out}_@ = \mathcal{L}$ and $\mathsf{init}_\Omega = \overline{\mathcal{L}}$, this directly extends $t_\mathcal{L}$ to an automaton homomorphism.

For bijectivity of these operations, note that by definition we have $\mathsf{out}_\Omega \circ o_{@,\mathcal{L}} = \mathcal{L}$ and $r_{\Omega,\overline{\mathcal{L}}} \circ \mathsf{init}_@ = \overline{\mathcal{L}}$. We thus only need to show $o_{@,\mathsf{out}_\Omega \circ t_\mathcal{L}} = t_\mathcal{L}$ and $r_{\Omega, t_\mathcal{L} \circ \mathsf{init}_@} = t_\mathcal{L}$, but these follow using the uniqueness properties of observability and reachability maps from the fact that $t_\mathcal{L}$ is an automaton homomorphism. □

Let us explore $t_\mathcal{L}$ for a language $\mathcal{L}\colon A^* \to 2$ while we verify that the corresponding $\overline{\mathcal{L}}\colon 1 \to 2^{A^*}$ is defined as expected.

Proposition 2.11. *Given a language $\mathcal{L}\colon A^* \to 2$, $t_\mathcal{L}\colon A^* \to 2^{A^*}$ and $\overline{\mathcal{L}}\colon 1 \to 2^{A^*}$ are*

$$t_\mathcal{L}(u)(v) = \mathcal{L}(uv) \qquad\qquad \overline{\mathcal{L}}(*)(u) = \mathcal{L}(u).$$

Proof. The dynamorphism $t_\mathcal{L}$ is obtained in the proof of Proposition 2.10 as $o_@$ (specifically, $o_{@,\mathcal{L}}$), which we recall being defined inductively:

$$\begin{aligned} o_@(u)(\varepsilon) &= \mathsf{out}_@(u) & \qquad o_@(u)(av) &= o_@(\delta_@(u)(a))(v) \\ &= \mathcal{L}(u) & &= o_@(ua)(v). \end{aligned}$$

A trivial proof by induction on the length of words v yields $t_{\mathcal{L}}(u)(v) = o_{@}(u)(v) = \mathcal{L}(uv)$. Furthermore,

$$\overline{\mathcal{L}}(*)(u) = (t_{\mathcal{L}} \circ \mathsf{init}_{@})(*)(u) = t_{\mathcal{L}}(\varepsilon)(u) = \mathcal{L}(u). \qquad \square$$

To keep things simple, we refer also in the abstract setting to a morphism $\mathcal{L}\colon @ \to Y$ as a *language.*

Definition 2.12 (Language)**.** A morphism $\mathcal{L}\colon @ \to Y$ is called a *language.* The corresponding dynamorphism $t_{\mathcal{L}}$ is the *total response* [11] of the language $\mathcal{L}$. We define the language $\mathcal{L}_Q$ accepted by an automaton Q as $\mathcal{L}_Q = \mathsf{out}_Q \circ r_Q$.

Next, we interpret the correspondence of Proposition 2.10 for an automaton.

Proposition 2.13. *Given an automaton Q, we have*

$$t_{\mathcal{L}_Q} = o_Q \circ r_Q \qquad\qquad \overline{\mathcal{L}_Q} = o_Q \circ \mathsf{init}_Q.$$

Proof. Taking $t_{\mathcal{L}} = o_Q \circ r_Q$, which is known to be a dynamorphism, we use the operations defined in the proof of Proposition 2.10 to find in the commutative diagrams below that the corresponding $\mathcal{L}$ is $\mathsf{out}_Q \circ r_Q = \mathcal{L}_Q$ and that $\overline{\mathcal{L}}$ is $o_Q \circ \mathsf{init}_Q$. By bijectivity of the construction, this completes the proof.

$$\begin{tikzcd}
& Q \arrow[d, "o_Q"] \arrow[dr, "\mathsf{out}_Q"] & \\
{@} \arrow[ur, "r_Q"] \arrow[r, "t_{\mathcal{L}}"] \arrow[rr, bend right, "\mathcal{L}"'] & \Omega \arrow[r, "\mathsf{out}_\Omega"] & Y
\end{tikzcd}
\qquad
\begin{tikzcd}
I \arrow[r, "\mathsf{init}_{@}"'] \arrow[rr, bend left, "\overline{\mathcal{L}}"] \arrow[dr, "\mathsf{init}_Q"'] & {@} \arrow[r, "t_{\mathcal{L}}"'] \arrow[d, "r_Q"] & \Omega \\
& Q \arrow[ur, "o_Q"'] &
\end{tikzcd}$$

□

Note that Proposition 2.13 equivalently says that r_Q and o_Q are automaton homomorphisms with respect to the automaton extensions of @ and Ω for the language $\mathcal{L}_Q$. To conclude this section, we show that automaton homomorphisms witness language equivalence of the participating automata.

Proposition 2.14. *For any two automata Q and R, if there exists an automaton homomorphism $f\colon Q \to R$, then $\mathcal{L}_R = \mathcal{L}_Q$.*

Proof. Since f is in particular an input system homomorphism, we have by initiality of @ that $r_R = f \circ r_Q$. Therefore, and using that f is an output system homomorphism,

$$\mathcal{L}_R = \mathsf{out}_R \circ r_R = \mathsf{out}_R \circ f \circ r_Q = \mathsf{out}_Q \circ r_Q = \mathcal{L}_Q. \qquad \square$$

2.4 A Minimal Realization

The usual construction due to Nerode [54] of a minimal automaton for a language $\mathcal{L}$ takes certain equivalence classes as the state space, but it can equivalently be seen as taking the image of the total response $t_{\mathcal{L}}$. Towards an abstract version of this construction, we introduce a specific type of *factorization systems*. For a more abstract treatment we commend the work of Adámek et al. [5].

Definition 2.15 (Factorization System)**.** A *factorization system* for a category $\mathbf{C}$ is a pair $(\mathcal{E}, \mathcal{M})$ of classes of morphisms in $\mathbf{C}$ such that

(F1) both $\mathcal{E}$ and $\mathcal{M}$ are closed under composition;
(F2) all morphisms in $\mathcal{E}$ are epis; all morphisms in $\mathcal{M}$ are monos;
(F3) if $g \circ f \in \mathcal{E}$, then $g \in \mathcal{E}$; if $g \circ f \in \mathcal{M}$, then $f \in \mathcal{M}$;
(F4) each morphism f in $\mathbf{C}$ can be decomposed as $f = m \circ e$, where $e \in \mathcal{E}$ and $m \in \mathcal{M}$; and
(F5) for each commutative square in $\mathbf{C}$ as below on the left,

$$\begin{tikzcd} U \arrow[r, two heads, "i"] \arrow[d, "f"'] & V \arrow[d, "g"] \\ W \arrow[r, tail, "j"] & X \end{tikzcd} \qquad\qquad \begin{tikzcd} U \arrow[r, two heads, "i"] \arrow[d, "f"'] & V \arrow[d, "g"] \arrow[dl, dotted, "d"'] \\ W \arrow[r, tail, "j"] & X \end{tikzcd} \tag{2.1}$$

where $i \in \mathcal{E}$ and $j \in \mathcal{M}$, there exists a diagonal d making the triangles on the right commute.

Note that both triangles commute if this is the case for one of them. For instance, if $d \circ i = f$, then $j \circ d \circ i = j \circ f = g \circ i$, from which

$j \circ d = g$ follows because i is an epi; for the same reason this diagonal d is necessarily unique. The diagonalization property is equivalent to the factorizations being unique up to unique isomorphism, and thus a factorization system intuitively provides a canonical image for each morphism. We only give a proof for the implication here [5, Proposition 14.4].

Proposition 2.16 (Uniqueness of Factorizations)**.** *If for any morphism $f \colon U \to V$ in a category with a factorization system $(\mathcal{E}, \mathcal{M})$ we have $f = m_1 \circ e_1 = m_2 \circ e_2$ for $e_1 \colon U \to W$ and $e_2 \colon U \to X$ and $\mathcal{E}$ and $m_1 \colon W \to V$ and $m_2 \colon X \to V$ in $\mathcal{M}$, then there exists a unique isomorphism $\phi \colon W \to X$ satisfying $\phi \circ e_1 = e_2$ and $m_1 = m_2 \circ \phi$.*

Proof. There are two commutative squares for which we can find unique diagonals:

$$\begin{tikzcd}
U \arrow[r, two heads, "e_1"] \arrow[d, "e_2"'] & W \arrow[d, "m_1"] \arrow[dl, dotted, "\phi"'] \\
X \arrow[r, tail, "m_2"] & V
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
U \arrow[r, two heads, "e_2"] \arrow[d, "e_1"'] & X \arrow[d, "m_2"] \arrow[dl, dotted, "\psi"'] \\
W \arrow[r, tail, "m_1"] & V
\end{tikzcd}$$

Then also

$$\begin{tikzcd}
U \arrow[rr, two heads, "e_1"] \arrow[dd, "e_1"'] \arrow[dr, "e_2"] & & W \arrow[dd, "m_1"] \arrow[dl, "\phi"'] \\
& X \arrow[dl, "\psi"'] \arrow[dr, "m_2"] & \\
W \arrow[rr, tail, "m_1"] & & V
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
U \arrow[rr, two heads, "e_2"] \arrow[dd, "e_2"'] \arrow[dr, "e_1"] & & X \arrow[dd, "m_2"] \arrow[dl, "\psi"'] \\
& W \arrow[dl, "\phi"'] \arrow[dr, "m_1"] & \\
X \arrow[rr, tail, "m_2"] & & V
\end{tikzcd}$$

commute, so by uniqueness of the diagonals we conclude $\psi \circ \phi = \mathsf{id}_W$ and $\phi \circ \psi = \mathsf{id}_X$. □

We will rely quite heavily on (F2), which may be interpreted as the system providing a notion of some sort of minimality rather than an even more vague notion of canonicity. The properties (F1) and (F3) become relevant only when learning is considered.

In the category of sets and functions, we take the factorization system

$$\text{(surjective functions, injective functions)},$$

which satisfies (F1) through (F3) as we have seen in Section 2.1. Before proceeding to the other properties, we first define define for any function $f\colon U \to V$ the sets

$$\begin{aligned} \mathsf{im}(f) &= \{f(u) \in V \mid u \in U\} \quad \text{and} \\ \mathsf{ker}(f) &= \{(u_1, u_2) \in U \times U \mid f(u_1) = f(u_2)\}, \end{aligned}$$

and give the following more elaborate lemma that will also be useful later.

Lemma 2.17. *Consider functions f and g that complete the respective diagrams below in* **Sets**.

$$\begin{tikzcd} C \arrow[d, dotted, "f"'] \arrow[dr, "x"] & \\ D \arrow[r, tail, "u"] & E \end{tikzcd} \qquad \begin{tikzcd} K \arrow[r, two heads, "v"] \arrow[dr, "y"'] & L \arrow[d, dotted, "g"] \\ & M \end{tikzcd}$$

The function f exists if and only if $\mathsf{im}(x) \subseteq \mathsf{im}(u)$; the function g exists if and only if $\mathsf{ker}(v) \subseteq \mathsf{ker}(y)$.

Proof. If f exists, then for any element $c \in C$, $f(c) \in D$ satisfies $u(f(c)) = x(c)$. Conversely, assume that for every $c \in C$ there exists a $d \in D$ such that $u(d) = x(c)$, and let $f(c)$ be that d. It follows immediately that $u(f(c)) = x(c)$.

If g exists, then for all $k_1, k_2 \in K$ with $v(k_1) = v(k_2)$ we have

$$y(k_1) = g(v(k_1)) = g(v(k_2)) = y(k_2).$$

Conversely, assume that for all $k_1, k_2 \in K$ with $v(k_1) = v(k_2)$ we have $y(k_1) = y(k_2)$. For each $l \in L$, pick any $k \in K$ that satisfies $v(k) = l$, which is possible since v is surjective, and define $g(l) = y(k)$. Consider $k' \in K$, define $l = v(k')$, and let k be the element of K such that $v(k) = l$ and $g(l) = y(k)$ as provided by the above definition. This gives us

$$g(v(k')) = g(l) = y(k),$$

and finally $y(k) = y(k')$ follows from our assumption after observing that $v(k) = l = v(k')$. □

Now note that a function $f\colon U \to V$ can be factorized as

$$\begin{tikzcd} U \arrow[r, two heads, "e"] \arrow[rr, bend right, "f"'] & \mathsf{im}(f) \arrow[r, tail, "m"] & V \end{tikzcd}, \quad \text{where} \quad \begin{array}{l} e(u) = f(u) \;\text{ and} \\ m(v) = v. \end{array}$$

We can readily see that e is surjective and that m is injective.

For each commuting square of functions as in (F5), the diagonal exists by Lemma 2.17 if $\mathsf{im}(g) \subseteq \mathsf{im}(j)$, which is seen from surjectivity of i:

$$\mathsf{im}(g) = \mathsf{im}(g \circ i) = \mathsf{im}(j \circ f) \subseteq \mathsf{im}(j).$$

On an abstract level, we assume that $\mathbf{B}$ has a factorization system $(\mathcal{E}, \mathcal{M})$ and that $\mathcal{D}$ preserves $\mathcal{M}$ (i.e., if $f \in \mathcal{M}$, then $\mathcal{D}f \in \mathcal{M}$). The latter implies that we have factorizations of automaton homomorphisms, which will lead to the identification of minimal automata.

Lemma 2.18. *If $h\colon U \to V$ in $\mathbf{Aut}$ is factorized as $m \circ e$ through an object M in $\mathbf{B}$, then M is an automaton such that m and e are automaton homomorphisms.*

Proof. We define init_M and out_M as the compositions indicated below on the left.

$$\begin{tikzcd} I \arrow[d, "\mathsf{init}_U"'] \arrow[dr, dotted, "\mathsf{init}_M"] & & Y \\ U \arrow[r, two heads, "e"] \arrow[rr, bend right, "h"'] & M \arrow[r, tail, "m"] \arrow[ur, dotted, "\mathsf{out}_M"] & V \arrow[u, "\mathsf{out}_V"'] \end{tikzcd} \qquad \begin{tikzcd} U \arrow[r, two heads, "e"] \arrow[d, "\delta_U"'] & M \arrow[d, "m"] \arrow[ddl, dotted, "\delta_M"'] \\ \mathcal{D}U \arrow[d, "\mathcal{D}e"'] & V \arrow[d, "\delta_V"] \\ \mathcal{D}M \arrow[r, tail, "\mathcal{D}m"'] & \mathcal{D}V \end{tikzcd} \qquad \begin{tikzcd} U \arrow[r, "e"] \arrow[d, "\delta_U"'] \arrow[dr, "h"'] & M \arrow[d, "m"] \\ \mathcal{D}U \arrow[d, "\mathcal{D}e"'] \arrow[dr, "\mathcal{D}h"] & V \arrow[d, "\delta_V"] \\ \mathcal{D}M \arrow[r, "\mathcal{D}m"'] & \mathcal{D}V \end{tikzcd}$$

The dynamics δ_M is obtained as the diagonal running through the rectangle in the middle. This requires commutativity thereof, which we derive on the right from $m \circ e = h$ being a dynamorphism.

It remains to show that m and e are automaton homomorphisms. By the definition of init_M above, e commutes with the initial states; by the definition of out_M, m commutes with the output maps. Moreover, the definition of the δ_M causes both e and m to be dynamorphisms. Thus, we only check $\mathsf{out}_U = \mathsf{out}_M \circ e$ and $\mathsf{init}_V = m \circ \mathsf{init}_M$ (the non-triangular regions below are homomorphism properties of h):

$$\begin{tikzcd}
& M \arrow[d, "m"] \arrow[dr, "\mathsf{out}_M"] & \\
U \arrow[ur, "e"] \arrow[r, "h"] \arrow[rr, bend right, "\mathsf{out}_U"'] & V \arrow[r, "\mathsf{out}_V"] & Y
\end{tikzcd}
\qquad
\begin{tikzcd}
I \arrow[r, "\mathsf{init}_U"'] \arrow[rr, bend left, "\mathsf{init}_V"] \arrow[dr, "\mathsf{init}_M"'] & U \arrow[r, "h"'] \arrow[d, "e"] & V \\
& M \arrow[ur, "m"'] &
\end{tikzcd}$$

□

Lemma 2.19. *If a commuting square through which a diagonal is obtained using the factorization system in* $\mathbf{B}$ *is a square of automaton homomorphisms, then the diagonal is also an automaton homomorphism.*

Proof. Consider a commuting square of automaton homomorphisms as below, and suppose we have obtained d as the diagonal in $\mathbf{B}$.

$$\begin{tikzcd}
U \arrow[r, two heads, "i"] \arrow[d, "f"'] & V \arrow[d, "g"] \arrow[dl, dotted, "d"'] \\
W \arrow[r, tail, "j"'] & X
\end{tikzcd}$$

We prove that d is an automaton homomorphism by showing

$$\begin{aligned} j \circ d \circ \mathsf{init}_V &= j \circ \mathsf{init}_W \\ \mathsf{out}_W \circ d \circ i &= \mathsf{out}_V \circ i \end{aligned} \qquad \delta_V \circ d \circ i = \mathcal{D}d \circ \delta_U \circ i,$$

from which the expected results follow using that j is a mono and i

an epi.

$$\begin{tikzcd}
I \arrow[rr, "\mathsf{init}_W"] \arrow[dd, "\mathsf{init}_V"'] \arrow[drr, bend right, "\mathsf{init}_X"] & & W \arrow[d, tail, "j"] \\
 & & X \\
V \arrow[urr, "g"] \arrow[rr, "d"'] & & W \arrow[u, tail, "j"']
\end{tikzcd}
\qquad
\begin{tikzcd}
V \arrow[r, "d"] & W \arrow[dd, "\mathsf{out}_W"] \\
U \arrow[u, two heads, "i"] \arrow[d, two heads, "i"'] \arrow[ur, "f"'] \arrow[dr, bend left, "\mathsf{out}_U"'] & \\
V \arrow[r, "\mathsf{out}_V"'] & Y
\end{tikzcd}
\qquad
\begin{tikzcd}
V \arrow[rr, "\delta_V"] & & \mathcal{D}V \arrow[dd, "\mathcal{D}d"] \\
U \arrow[u, two heads, "i"] \arrow[d, two heads, "i"'] \arrow[r, "\delta_U"] \arrow[dr, "f"] & \mathcal{D}U \arrow[ur, "\mathcal{D}i"] \arrow[dr, "\mathcal{D}f"] & \\
V \arrow[r, "d"'] & W \arrow[r, "\delta_W"'] & \mathcal{D}W
\end{tikzcd}$$

□

Proposition 2.20 (Automaton Factorizations)**.** *The category* **Aut** *has a factorization system inherited from* **B***. Specifically, it is given by*

$$(\{e \in \mathcal{E} \mid e \textit{ is an automaton homomorphism}\},$$
$$\{m \in \mathcal{M} \mid m \textit{ is an automaton homomorphism}\}).$$

Proof. Because automaton homomorphisms that are epic (monic) in **B** are also epic (monic) in **Aut**, we see that the properties (F1) through (F3) are inherited directly. Furthermore, (F4) is provided by Lemma 2.18, and with Lemma 2.19 we conclude that (F5) also holds. □

Before constructing a minimal automaton, we need to define minimality. Using the following definition, the uniqueness property of factorizations is automatically transferred to minimal automata.

Definition 2.21 (Minimality [11])**.** An input system U is called *reachable* if its reachability map r_U is in $\mathcal{E}$. An output system V is called *observable* if its observability map o_V is in $\mathcal{M}$. We call an automaton *minimal* if it is both reachable and observable.

For DA this coincides with the usual notion of minimality: a DA is minimal if all of its states can be reached from the initial state (reachability) and no two states accept the same language (observability).

Consider the DA in Figure 2.3. It is neither reachable—state q_3 cannot be reached—nor observable—states q_1 and q_2 accept the

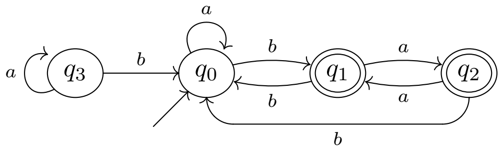


Figure 2.3: DA that is neither reachable nor observable

same language. If we remove q_3 and merge q_1 with q_2, we arrive at a minimal DA that is isomorphic to the one shown in Figure 2.2.

Considering the fact that the total response is an automaton homomorphism that can be factorized, it is now a trivial task to identify a minimal automaton for a given language.

Proposition 2.22 (Minimal Realization)**.** *There exists a unique (up to automaton isomorphism) minimal automaton M satisfying $\mathcal{L}_M = \mathcal{L}$.*

Proof. We have seen in Proposition 2.10 that $t_{\mathcal{L}}$ is an automaton homomorphism, so using Proposition 2.20 we factorize $t_{\mathcal{L}} = o_M \circ r_M$ through an automaton M. Note that o_M and r_M in this definition are automaton homomorphism, so by uniqueness of the reachability and observability maps this validates our suggestive naming. Furthermore, since r_M is defined to be in $\mathcal{E}$ and o_M is defined to be in $\mathcal{M}$, the automaton M is minimal. By Proposition 2.13,

$$t_{\mathcal{L}_M} = o_M \circ r_M = t_{\mathcal{L}},$$

and hence we have $\mathcal{L}_M = \mathcal{L}$. Uniqueness follows immediately from the uniqueness of factorizations. □

Let us spell out the resulting automaton: $t_{\mathcal{L}} = o_M \circ r_M$ is a factorization through M in **Aut**, which is just a factorization in **B** with the additional definitions $\mathsf{init}_M = r_M \circ \mathsf{init}_@$, $\mathsf{out}_M = \mathsf{out}_\Omega \circ o_M$, and the dynamics δ_M being obtained using diagonalization:

$$\begin{tikzcd}
@ \arrow[r, two heads, "r_M"] \arrow[rr, bend right, "t_{\mathcal{L}}"'] & M \arrow[r, tail, "o_M"] & \Omega
\end{tikzcd}
\qquad
\begin{tikzcd}
I \arrow[d, "\mathsf{init}_@"'] \arrow[dr, dotted, two heads, "\mathsf{init}_M"] & & Y \\
@ \arrow[r, two heads, "r_M"'] \arrow[d, "\delta_@"'] & M \arrow[r, "o_M"'] \arrow[d, dotted, "\delta_M"] \arrow[ur, dotted, "\mathsf{out}_M"] & \Omega \arrow[u, "\mathsf{out}_\Omega"'] \arrow[d, "\delta_\Omega"] \\
\mathcal{D}@ \arrow[r, "\mathcal{D}r_M"] & \mathcal{D}M \arrow[r, tail, "\mathcal{D}o_M"] & \mathcal{D}\Omega
\end{tikzcd}$$

In the specific case of DA, this results in the definition

$$M = \{t_{\mathcal{L}}(u) \mid u \in A^*\} \qquad \mathsf{init}_M(*) = t_{\mathcal{L}}(\varepsilon)$$
$$\delta_M(t_{\mathcal{L}}(u))(a) = t_{\mathcal{L}}(ua) \qquad \mathsf{out}_M(t_{\mathcal{L}}(u)) = \mathcal{L}(u),$$

where by Proposition 2.11 we can think of $t_{\mathcal{L}}(u)$ as the residual language after reading the word $u \in A^*$. Indeed, with our definition of factorizing functions in **Sets**, o_M is just an inclusion.

For instance, recall that the automaton Q in Figure 2.2 is the minimal realization for the language $\mathcal{L}$ of words over $\{a, b\}$ containing an odd number of b's. The states of the isomorphic automaton M are given by the languages accepted by the states of Q—the language of words containing an odd number of b's for q_0 and the language of words containing an even number of b's for q_1.

3 Automata Learning

In *active* automata learning, a language $\mathcal{L}$ is known in that we may ask *membership queries*, where we submit a word $u \in A^*$ and retrieve whether $u \in \mathcal{L}$. However, no complete description of $\mathcal{L}$ is given; the goal of the learner is to find an automaton H such that $\mathcal{L}_H = \mathcal{L}$.

We do not consider *passive* learning, where membership queries are not allowed, but a *sample*, a partial description of $\mathcal{L}$, is given. There are some very strong complexity results against finding the minimal DA consistent with the sample, most notably by Pitt and Warmuth [59], who show that finding even an approximately minimal DA is NP-hard.

Surprisingly, the popular active learning theory also traces back to the paper by Arbib and Zeiger [14], who, apart from motivating an abstract study of automata, generalized an identification procedure for linear systems developed by Ho [37]. The idea is that the conceptual construction of the minimal automaton due to Nerode [54]—the factorization of $t_{\mathcal{L}}$, as we have formulated it—can be replaced by limiting the total response from $A^* \to 2^{A^*}$ to $A^{\leq n} \to 2^{A^{\leq m}}$ for $n, m \in \mathbb{N}$ such that for each state q in the minimal automaton there is a word $u \in A^{\leq n} = \{v \in A^* \mid |v| \leq n\}$ that reaches q ($r_M(u) = q$) and for all states $q_1, q_2 \in M$ there is a word $v \in A^{\leq m}$ that distinguishes q_1 from q_2 ($o_M(q_1)(v) \neq o_M(q_2)(v)$).

This concept was optimized by Gold [36], who additionally notes that rather than assuming a given upper bound on the number of states of M and using it to choose sufficiently large n and m, one may learn *indefinitely* by iteratively increasing these parameters so that the automaton is learned *in the limit*.

Procedures introduced by Gold to still produce an automaton at incomplete stages were picked up by Angluin [7], who provides a polynomial time algorithm using a quite different assumption: she assumes that any automaton can be tested for equivalence with M, yielding a *counterexample* in the case of a negative result. These enquiries are referred to as *equivalence queries*. Equivalence queries are used to produce a sequence of hypothesis automata, where each next hypothesis has more states than the previous one and the last automaton in the sequence is M.

From a practical perspective, the assumption of being able to determine equivalence with M amounts to abstracting away from the hardest part of the problem. In a way, the automata learning problem is as hard as the problem of answering equivalence queries. If we can answer equivalence queries, then Angluin's algorithm can be applied to learn M. The algorithm asks a number of equivalence queries that is linear in the size of M, and, abstracting from those and the membership queries, the algorithm itself runs in polynomial time (in the sizes of M and A), provided that counterexamples are of polynomial length. Conversely, if we can learn an automaton equivalent to a certain black box, we can determine equivalence of any automaton with that black box by using the usual automata theoretical methods to determine equivalence. These methods can also be polynomial, provided that the learning algorithm produces an automaton of polynomial size.

One way to realize equivalence queries is by using a *conformance testing* algorithm. In general, conformance testing consists in testing conformance of some system with a given specification by means of experimentation [65]. We confine ourselves here to the case where the specification is an automaton and conformance amounts to equivalence with that automaton. A well-known example of a conformance tester in this sense is the W-method due to Vasilevskii [67] and Chow [30]. Berg et al. [19] show a correspondence between the learning algorithm of Angluin and the W-method that is much stronger than the reductions mentioned above. We will explain part of this in Section 4.1. That is, we will explain the W-method using concepts and results that will be developed in the present section for learning.

One could wonder if the conformance tester could not be integrated into the learning algorithm, yielding a likely more optimized version of the algorithm by Gold. The approach of Angluin, however, enjoys the usual advantage of a modular design: if conformance testing methods, or any other realizations of equivalence queries, already exist for the type of automata under consideration, they can easily be plugged into the learning algorithm, and one method can be replaced by another without much effort. We can even weaken the correctness requirement for the algorithm and adopt a stochastic setting, where the conformance tester is implemented by drawing tests from A^* at random, assuming some probability distribution. This was already suggested by Angluin [7], who places the resulting algorithm in the probably approximately correct (PAC) learning framework due to Valiant [66].

The remainder of this section is organized as follows. We develop for our abstract setting the concept of an approximation of the total response in Section 3.1, and we show that it encompasses different structures proposed for Angluin-style learning. Section 3.2 subsequently further explains and generalizes the observations of Gold, and we turn to Angluin's concrete algorithm in Section 3.3. Section 3.4 concludes with related and future work.

3.1 Approximating the Total Response

We can think of $t_{\mathcal{L}}$ as an infinite table with rows and columns such that a cell in a row with label $u \in A^*$ and a column with label $v \in A^*$ has the value $t_{\mathcal{L}}(u)(v) = \mathcal{L}(uv)$. That is, each cell records whether the corresponding row label concatenated with the column label is in the language. Since the automaton M is the factorization (image) of $t_{\mathcal{L}}$, its states are given by the different rows in this table. We will assume that M is a finite set so that the number of distinct rows is finite, but note that we face two problems in naively computing the image of $t_{\mathcal{L}}$: first, we cannot iterate over all of A^*; second, we cannot compare or even represent arbitrary elements of 2^{A^*}. To overcome these problems, we study approximations of the total response.

Definition 3.1 (Approximation)**.** An *approximation* is a tuple (S, P, σ, π) comprising objects S and P and morphisms $\sigma \colon S \to @$ and $\pi \colon \Omega \to P$ in $\mathbf{B}$. We call the composition ξ defined below the *approximated response* associated with such an approximation.

$$\begin{tikzcd} S \arrow[r, "\sigma"] \arrow[rrr, dotted, bend right, "\xi"'] & @ \arrow[r, "t_{\mathcal{L}}"] & \Omega \arrow[r, "\pi"] & P \end{tikzcd}$$

One can think of σ as representing an inclusion $S \subseteq A^*$ and of π as partitioning the set of languages 2^{A^*} into partitions named by the elements of P. In general, however, we do not require σ to be monic or π to be epic.

Arguably the simplest kind of approximation for DA is the one underlying the *observation tables* [36, 7] that will be introduced later. Here one takes finite sets $S, E \subseteq A^*$ and defines $P = 2^E$ along with $\sigma_S(s) = s$ for each $s \in S$ and $\pi_E(l)(e) = l(e)$ for $e \in E$. The elements of S are referred to as *access strings* while the elements of E are called *experiments*. We define the *observation table approximation*

$$\mathsf{T}(S, E) = (S, 2^E, \sigma_S, \pi_E)$$

for convenience. The infinite table $t_{\mathcal{L}}$ is now limited into the finite table ξ: σ_S selects a finite set of rows while π_E picks a finite number of columns. That is, the approximated response is as follows.

Proposition 3.2. *For $\mathsf{T}(S, E)$, the function $\xi\colon S \to 2^E$ is given by*

$$\xi(s)(e) = \mathcal{L}(se).$$

Proof. Because σ_S and π_E simply formalize inclusions, this ultimately follows just from the definition of $t_{\mathcal{L}}$ obtained in Proposition 2.11:

$$\begin{aligned} \xi(s)(e) &= (\pi_E \circ t_{\mathcal{L}} \circ \sigma_S)(s)(e) = (\pi_E \circ t_{\mathcal{L}})(s)(e) \\ &= \pi_E(t_{\mathcal{L}}(s))(e) = t_{\mathcal{L}}(s)(e) = \mathcal{L}(se). \end{aligned} \qquad \square$$

An example of an approximated response for an observation table approximation is visualized in Table 3.1.

	a	b	ba	bb
a	0	1	1	0
b	1	0	0	1
bb	0	1	1	0

Table 3.1: Example approximated response for the approximation $\mathsf{T}(\{a, b, bb\}, \{a, b, ba, bb\})$ targeting the language of the DA in Figure 2.2—the language of words over $\{a, b\}$ containing an odd number of b's

We fix an arbitrary approximation (S, P, σ, π). The goal of the learning algorithms that we consider is to construct M (up to isomorphism) from the approximated response ξ by factorizing the latter. Thus, let us factorize $\xi = m \circ e$ through an object H in $\mathbf{B}$.

$$\begin{tikzcd} S \arrow[r, two heads, "e"] \arrow[rr, bend right, "\xi"'] & H \arrow[r, tail, "m"] & P \end{tikzcd}$$

By the definition of ξ and M,

$$\xi = \pi \circ t_{\mathcal{L}} \circ \sigma = \pi \circ o_M \circ r_M \circ \sigma,$$

so the following notion of a *complete* approximation arises naturally as a sufficient condition to guarantee a successful factorization of ξ.

Definition 3.3. The approximation *reaches* M if $r_M \circ \sigma \in \mathcal{E}$; it *observes* M if $\pi \circ o_M \in \mathcal{M}$. The approximation is called *complete* if it both reaches and observes M.

If the approximation is complete, we immediately obtain the state space of M up to isomorphism by factorizing ξ. Conceptually, we can even turn this factorization into an automaton isomorphic to M, but the proof is not very informative.

Proposition 3.4. *If the approximation is complete, the approximated response factorizes through an automaton isomorphic to M.*

Proof. As the approximation is complete, there is by the uniqueness of factorizations an isomorphism $\phi\colon M \to H$ in $\mathbf{B}$; denote by ψ its inverse. Now define an automaton structure on H by the compositions indicated below.

$$\begin{tikzcd}
I \arrow[d, "\mathsf{init}_M"'] \arrow[dr, dotted, "\mathsf{init}_H"] & & Y \\
M \arrow[r, "\phi"] & H \arrow[ur, dotted, "\mathsf{out}_H"] \arrow[r, "\psi"] \arrow[rrr, dotted, bend right, "\delta_H"'] & M \arrow[u, "\mathsf{out}_M"'] \arrow[r, "\delta_M"] & \mathcal{D}M \arrow[r, "\mathcal{D}\phi"] & \mathcal{D}H
\end{tikzcd}$$

This construction directly gives us the equations $\phi \circ \mathsf{init}_M = \mathsf{init}_H$ and $\mathsf{out}_M \circ \psi = \mathsf{out}_H$ needed to make ϕ and ψ automaton homomorphisms. The remainder—$\psi \circ \mathsf{init}_H = \mathsf{init}_M$, $\mathsf{out}_H \circ \phi = \mathsf{out}_M$, $\delta_H \circ \phi = \mathcal{D}\phi \circ \delta_M$, and $\delta_M \circ \psi = \mathcal{D}\psi \circ \delta_H$—is proven easily by observing that ϕ and ψ are inverse to each other.

$$\begin{tikzcd}
 & I \arrow[dl, "\mathsf{init}_M"'] \arrow[d, "\mathsf{init}_H"] \\
M \arrow[r, "\phi"] \arrow[dr, "\mathsf{id}"'] & H \arrow[d, "\psi"] \\
 & M
\end{tikzcd}
\qquad
\begin{tikzcd}
M \arrow[d, "\phi"'] \arrow[dr, "\mathsf{id}"] & \\
H \arrow[r, "\psi"] \arrow[d, "\mathsf{out}_H"'] & M \arrow[dl, "\mathsf{out}_M"] \\
Y &
\end{tikzcd}
\qquad
\begin{tikzcd}
M \arrow[d, "\phi"'] \arrow[dr, "\mathsf{id}"] & \\
H \arrow[r, "\psi"] \arrow[ddr, "\delta_H"'] & M \arrow[d, "\delta_M"] \\
 & \mathcal{D}M \arrow[d, "\mathcal{D}\phi"] \\
 & \mathcal{D}H
\end{tikzcd}
\qquad
\begin{tikzcd}
H \arrow[r, "\psi"] \arrow[ddr, bend right, "\delta_H"'] & M \arrow[d, "\delta_M"'] \\
 & \mathcal{D}M \arrow[d, "\mathcal{D}\phi"'] \arrow[dd, bend left, "\mathcal{D}\mathsf{id}"] \\
 & \mathcal{D}H \arrow[d, "\mathcal{D}\psi"'] \\
 & \mathcal{D}M
\end{tikzcd}$$

This shows that they are automaton isomorphisms. □

Thus, H is the minimal automaton satisfying $t_H = t_{\mathcal{L}}$. However, its structure is defined in terms of the structure of M; we wish to rephrase the construction independent of the unknown M.

A crucial observation is that we did not yet exploit any of the additional properties of the isomorphisms ϕ and ψ. They are obtained by the uniqueness of factorizations (Proposition 2.16) in **B** as the diagonals making the triangles in the diagrams below commute.

$$\begin{tikzcd}
S \arrow[r, two heads, "r_M \circ \sigma"] \arrow[dd, "e"'] & M \arrow[d, "o_M"] \arrow[ddl, dotted, "\phi"'] \\
& \Omega \arrow[d, "\pi"] \\
H \arrow[r, tail, "m"] & P
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, two heads, "e"] \arrow[d, "\sigma"'] & H \arrow[dd, "m"] \arrow[ddl, dotted, "\psi"'] \\
@ \arrow[d, "r_M"'] & \\
M \arrow[r, tail, "\pi \circ o_M"] & P
\end{tikzcd}$$

Now note what happens if we compose the structure of H with the monos m and $\mathcal{D}m$ and if we precompose with the epi e:

$$\begin{tikzcd}
I \arrow[d, "\mathsf{init}_M"] \arrow[dr, bend left, "\mathsf{init}_H"] \arrow[dd, bend right=60, "\overline{\mathcal{L}}"'] & \\
M \arrow[r, "\phi"'] \arrow[d, "o_M"] & H \arrow[d, tail, "m"] \\
\Omega \arrow[r, "\pi"] & P
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, "\sigma"] \arrow[d, two heads, "e"'] & @ \arrow[d, "r_M"'] \arrow[dd, bend left=60, "\mathcal{L}"] \\
H \arrow[r, "\psi"] \arrow[dr, bend right, "\mathsf{out}_H"'] & M \arrow[d, "\mathsf{out}_M"'] \\
& Y
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, "\sigma"] \arrow[d, two heads, "e"'] & @ \arrow[d, "r_M"] \arrow[dr, bend left, "t_{\mathcal{L}}"] & \\
H \arrow[r, "\psi"] \arrow[ddr, bend right, "\delta_H"'] & M \arrow[r, "o_M"] \arrow[d, "\delta_M"] & \Omega \arrow[d, "\delta_\Omega"] \\
& \mathcal{D}M \arrow[r, "\mathcal{D}(o_M)"] \arrow[d, "\mathcal{D}\phi"] & \mathcal{D}\Omega \arrow[d, "\mathcal{D}\pi"] \\
& \mathcal{D}H \arrow[r, tail, "\mathcal{D}m"] & \mathcal{D}P
\end{tikzcd}$$

Because m is a mono, init_H is actually the unique morphism satisfying $m \circ \mathsf{init}_H = \pi \circ \overline{\mathcal{L}}$, the latter of which is independent of M. Similarly, out_H is unique in satisfying $\mathsf{out}_H \circ e = \mathcal{L} \circ \sigma$, and δ_H uniquely satisfies $\mathcal{D}m \circ \delta_H \circ e = \mathcal{D}\pi \circ \delta_\Omega \circ t_{\mathcal{L}} \circ \sigma$. Thus, we define the morphisms

$$\begin{array}{lll}
\xi_I \colon I \to P & \xi_Y \colon S \to Y & \xi_\delta \colon S \to \mathcal{D}P \\
\xi_I = \pi \circ \overline{\mathcal{L}} & \xi_Y = \mathcal{L} \circ \sigma & \xi_\delta = \mathcal{D}\pi \circ \delta_\Omega \circ t_{\mathcal{L}} \circ \sigma
\end{array}$$

to arrive at the following definition.

Definition 3.5 (Hypothesis). The approximation is *initialized* (*responsive*) if there exists a morphism init_H (out_H), making the diagram below on the left (middle) commute; it is *dynamical* if there exists a morphism δ_H making the diagram on the right commute.

$$\begin{tikzcd}
I \arrow[d, dotted, "\mathsf{init}_H"'] \arrow[dr, "\xi_I"] & \\
H \arrow[r, tail, "m"] & P
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, two heads, "e"] \arrow[dr, "\xi_Y"'] & H \arrow[d, dotted, "\mathsf{out}_H"] \\
& Y
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, two heads, "e"] \arrow[rrr, bend right, "\xi_\delta"'] & H \arrow[r, dotted, "\delta_H"] & \mathcal{D}H \arrow[r, tail, "\mathcal{D}m"] & \mathcal{D}P
\end{tikzcd}$$

Whenever the approximation is initialized, responsive, and dynamical, we call H the *hypothesis*.

The construction does not refer to the structure of M, and interestingly it also does not refer to the isomorphisms ϕ and ψ. A complete approximation is thus not required for this to work. Importantly, the automaton created in Proposition 3.4 does coincide with the hypothesis whenever the approximation is complete.

Concretely, in the base category **Sets** the hypothesis is obtained as

$$\begin{aligned}
H &= \{\xi(s) \mid s \in S\} & \qquad \mathsf{init}_H(i) &= \xi_I(i) \\
\delta_H(\xi(s)) &= \xi_\delta(s) & \qquad \mathsf{out}_H(\xi(s)) &= \xi_Y(s).
\end{aligned}$$

Well-definedness of these functions corresponds precisely to the abstract properties in Definition 3.5.

To get some feeling for this definition, we first evaluate ξ_I, ξ_Y, and ξ_δ for an observation table approximation.

Proposition 3.6. *For $\mathsf{T}(S, E)$, the functions ξ_I, ξ_Y, and ξ_δ are*

$$\begin{array}{ccc}
\xi_I \colon 1 \to 2^E & \xi_Y \colon S \to 2 & \xi_\delta \colon S \to (2^E)^A \\
\xi_I(*)(e) = \mathcal{L}(e) & \xi_Y(s) = \mathcal{L}(s) & \xi_\delta(s)(a)(e) = \mathcal{L}(sae).
\end{array}$$

Proof. The calculation for ξ_I and ξ_Y is quite trivial,

$$\begin{aligned}
&\xi_I(*)(e) = \pi_E(\overline{\mathcal{L}}(*))(e) = \overline{\mathcal{L}}(*)(e) = \mathcal{L}(e) \text{ and} \\
&\quad \xi_Y(s) = \mathcal{L}(\sigma_S(s)) = \mathcal{L}(s),
\end{aligned}$$

but ξ_δ needs some more work. Specifically, we need the definitions of $(-)^A$ and δ_Ω to see that

$$\begin{aligned}\xi_\delta(s)(a)(e) &= \pi_E^A(\delta_\Omega(t_{\mathcal{L}}(\sigma_S(s))))(a)(e) = \pi_E^A(\delta_\Omega(t_{\mathcal{L}}(s)))(a)(e)\\ &= (\pi_E \circ \delta_\Omega(t_{\mathcal{L}}(s)))(a)(e) = \pi_E(\delta_\Omega(t_{\mathcal{L}}(s))(a))(e)\\ &= \delta_\Omega(t_{\mathcal{L}}(s))(a)(e) = t_{\mathcal{L}}(s)(ae) = \mathcal{L}(sae). \qquad \square\end{aligned}$$

This allows us to interpret more concretely the properties that turn H into a suitable automaton.

Proposition 3.7. *In the base category* **Sets***, the approximation is initialized if and only if for each $i \in I$ there exists an $s \in S$ such that $\xi(s) = \xi_I(i)$; it is responsive if and only if for all $s_1, s_2 \in S$ with $\xi(s_1) = \xi(s_2)$ we have $\xi_Y(s_1) = \xi_Y(s_2)$.*

Proof. The first result follows from Lemma 2.17 by noting that m being an inclusion $H \hookrightarrow P$ satisfies $\mathsf{im}(m) = H = \mathsf{im}(\xi)$. For the second part, note that $e(s_1) = e(s_2)$ implies

$$\xi(s_1) = m(e(s_1)) = m(e(s_2)) = \xi(s_2),$$

and because m is injective we also have that $\xi(s_1) = \xi(s_2)$ implies $e(s_1) = e(s_2)$. That is, $\mathsf{ker}(e) = \mathsf{ker}(\xi)$. Again, the claim follows directly from Lemma 2.17. □

Proposition 3.8. *An approximation $\mathsf{T}(S, E)$ is initialized if and only if there exists an $s \in S$ such that $\mathcal{L}(se) = \mathcal{L}(e)$ for every $e \in E$; it is responsive if for all $s_1, s_2 \in S$ that satisfy $\mathcal{L}(s_1e) = \mathcal{L}(s_2e)$ for each $e \in E$ we have $\mathcal{L}(s_1) = \mathcal{L}(s_2)$.*

Proof. By Proposition 3.7 $\mathsf{T}(S, E)$ is initialized if and only if there is an $s \in S$ such that $\xi(s) = \xi_I(*)$. We are done by referring to the calculations of Proposition 3.2 and Proposition 3.6, which also directly derive the second claim from Proposition 3.7. □

Corollary 3.9. *An approximation $\mathsf{T}(S, E)$ is initialized if $\varepsilon \in S$; it is responsive if $\varepsilon \in E$.*

Note that in this corollary the conditions are strictly unnecessary. For example, the approximation underlying Table 3.1 does not include the empty word in either S or E, but it is still initialized because the row labeled by the word a equals the row for the empty word, if it would have been included in the table; and it is responsive, since the rows a and bb would still be equal if the table would be extended with a column for the empty word.

Dynamism can be split into two properties, which we name after properties originally defined by Angluin [7]. A first abstract reformulation of these was given by Jacobs and Silva [46].

Definition 3.10 (Closedness and Consistency)**.** We say that the approximation is *closed* (*consistent*) if there exists a morphism close (cons) making the bottom left (top right) triangle below commute.

$$\begin{tikzcd}
S \arrow[r, two heads, "e"] \arrow[d, dotted, "\mathsf{close}"'] \arrow[dr, "\xi_\delta"] & H \arrow[d, dotted, "\mathsf{cons}"] \\
\mathcal{D}H \arrow[r, tail, "\mathcal{D}m"] & \mathcal{D}P
\end{tikzcd}$$

Proposition 3.11. *The approximation is closed and consistent if and only if it is dynamical.*

Proof. First assume that the approximation is dynamical. Define close and cons as the compositions below on the left, so that we have

$$\begin{tikzcd}
S \arrow[r, two heads, "e"] \arrow[d, dotted, "\mathsf{close}"'] & H \arrow[d, dotted, "\mathsf{cons}"] \arrow[dl, "\delta_H"'] \\
\mathcal{D}H \arrow[r, tail, "\mathcal{D}m"] & \mathcal{D}P
\end{tikzcd}
\qquad
\begin{gathered}
\mathcal{D}m \circ \mathsf{close} = \mathcal{D}m \circ \delta_H \circ e = \xi_\delta \;\text{ and}\\
\mathsf{cons} \circ e = \mathcal{D}m \circ \delta_H \circ e = \xi_\delta.
\end{gathered}$$

Conversely, assume the approximation is closed and consistent. We obtain δ_H below on the left as a diagonal by commutativity of the square derived in the middle from closedness and consistency, and on the right we conclude that this definition satisfies dynamism.

$$\begin{tikzcd}
S \arrow[r, two heads, "e"] \arrow[d, "\mathsf{close}"'] & H \arrow[d, "\mathsf{cons}"] \arrow[dl, dotted, "\delta_H"'] \\
\mathcal{D}H \arrow[r, tail, "\mathcal{D}m"] & \mathcal{D}P
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, "e"] \arrow[d, "\mathsf{close}"'] \arrow[dr, "\xi_\delta"] & H \arrow[d, "\mathsf{cons}"] \\
\mathcal{D}H \arrow[r, "\mathcal{D}m"] & \mathcal{D}P
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[d, "e"'] \arrow[dr, "\mathsf{close}"] \arrow[drr, bend left, "\xi_\delta"] & & \\
H \arrow[r, "\delta_H"] & \mathcal{D}H \arrow[r, "\mathcal{D}m"] & \mathcal{D}P
\end{tikzcd}$$

□

Finally, we give concrete interpretations of closedness and consistency.

Proposition 3.12. *For the endofunctor $\mathcal{D} = (-)^A$ on the base category* **Sets***, the approximation is closed if and only if for all $s \in S$ and $a \in A$ we can find an $s' \in S$ that satisfies $\xi(s') = \xi_\delta(s)(a)$; it is consistent if and only if for all $s_1, s_2 \in S$ such that $\xi(s_1) = \xi(s_2)$ we have $\xi_\delta(s_1) = \xi_\delta(s_2)$.*

Proof. According to Lemma 2.17, $\mathsf{T}(S, E)$ is closed precisely if for each $s \in S$ there is a function $f\colon A \to H$ such that

$$\xi_\delta(s) = m^A(f) = m \circ f.$$

Thus, for each $a \in A$, we must have $\xi_\delta(s)(a) = m(f(a))$. The first result follows by noting that $m \circ e = \xi$ and that $f(a)$ is just $e(s')$ for some $s' \in S$ because e is surjective. As for the second part, we have already shown in the proof of Proposition 3.7 that $\mathsf{ker}(e) = \mathsf{ker}(\xi)$. □

Corollary 3.13. *An approximation $\mathsf{T}(S, E)$ is closed if and only if for all $s \in S$ and $a \in A$ we can find an $s' \in S$ that satisfies $\mathcal{L}(s'e) = \mathcal{L}(sae)$ for all $e \in E$; it is consistent if and only if for all $s_1, s_2 \in S$ such that $\mathcal{L}(s_1 e) = \mathcal{L}(s_2 e)$ for each $e \in E$ we also have $\mathcal{L}(s_1 ae) = \mathcal{L}(s_2 ae)$ for all $a \in A$ and $e \in E$.*

In summary, to determine if the hypothesis can be constructed from the approximation and to eventually construct it, we need to be aware of the following set of morphisms.

Definition 3.14 (Observation Structure)**.** The *observation structure* associated with the approximation is given by the following morphisms:

$$\begin{array}{llllll}
\xi & : & S \to P & : & \pi \circ t_{\mathcal{L}} \circ \sigma \\
\xi_I & : & I \to P & : & \pi \circ \overline{\mathcal{L}} \\
\xi_Y & : & S \to Y & : & \mathcal{L} \circ \sigma \\
\xi_\delta & : & S \to \mathcal{D}P & : & \mathcal{D}\pi \circ \delta_\Omega \circ t_{\mathcal{L}} \circ \sigma.
\end{array}$$

For practical reasons, we assume at this point the sets A and M to be finite. In active learning of DA, representations of appropriate observation structures are maintained using *membership queries*, where one submits a word $u \in A^*$ and receives the value of $\mathcal{L}(u)$. Importantly, this means we can think about extending approximations to make them initialized, responsive, and dynamical. Termination of such procedures relies on finiteness of the automaton M: the size of $\mathsf{im}(r_M \circ \sigma)$ can never exceed the size of M, nor can the size of $\mathsf{im}(\pi \circ o_M)$. Note that these functions are not known to us, so we cannot see these measures evolve while executing the algorithm. However, we do know their composition $\pi \circ o_M \circ r_M \circ \sigma = \pi \circ t_{\mathcal{L}} \circ \sigma = \xi$.

Remember that the original construction in Proposition 3.4 coincides with our more general hypothesis construction in the case of a complete approximation. Thus, we have the following.

Proposition 3.15. *If the approximation is complete, then the hypothesis is isomorphic to M.*

An actual proof will be given for the more general Theorem 4.10. The following result introduces the measure that is usually used to assess termination of active automata learning algorithms.

Proposition 3.16. *In* **Sets***, the approximation is complete if and only if $|H| = |M|$.*

Proof. If the approximation is complete, then by Proposition 3.15 we have $|H| = |M|$. Conversely, assume $|H| = |M|$. Since H is the image of ξ and $\xi = \pi \circ o_M \circ r_M \circ \sigma$, we have

$$|\operatorname{im}(r_M \circ \sigma)| \geq |M| \qquad\qquad |\operatorname{im}(\pi \circ o_M)| \geq |M|.$$

Because M is the codomain of $r_M \circ \sigma$ and the domain of $\pi \circ o_M$, we must have

$$|\operatorname{im}(r_M \circ \sigma)| = |M| \qquad\qquad |\operatorname{im}(\pi \circ o_M)| = |M|.$$

Therefore, $r_M \circ \sigma$ is surjective and $\pi \circ o_M$ is injective—the approximation is complete. □

Note that we always have $|H| \leq |M|$ because $H = \operatorname{im}(\pi \circ t_{\mathcal{L}} \circ \sigma)$ and $M = \operatorname{im}(t_{\mathcal{L}})$. Thus, to ensure termination we have to be able to increase $|H|$ until $|H| = |M|$.

3.1.1 Observation Tables

We can now fully explain the observation table as presented by Angluin [7]. Recall that the observation structure for such an approximation is as follows:

$$\begin{aligned} \xi &\colon S \to 2^E & \xi(s)(e) &= \mathcal{L}(se) \\ \xi_I &\colon 1 \to 2^E & \xi_I(*)(e) &= \mathcal{L}(e) \\ \xi_Y &\colon S \to 2 & \xi_Y(s) &= \mathcal{L}(s) \\ \xi_\delta &\colon S \to (2^E)^A & \xi_\delta(s)(a)(e) &= \mathcal{L}(sae). \end{aligned}$$

Angluin manages to represent these in a single table. To achieve this, she requires S and E to contain the empty word, so that by Corollary 3.9 her approximation is always initialized and responsive. She then splits the rows of her table into two parts. The upper part simply represents ξ, while the rows of the lower part are given by $\xi_\delta(s)(a)$ for $s \in S$ and $a \in A$ such that the word sa, which serves as the row label, is not already among the labels in the upper part. Note that if $sa \in S$, then $\xi(sa) = \xi_\delta(s)(a)$ so that ξ_δ is completely represented in the table. An example can be found in Table 3.2a.

	ε
ε	0
a	0
b	0
aa	0
ab	1

(a) Example table

	ε	b
ε	0	0
a	0	1
ab	1	0
b	0	0
aa	0	1
aba	1	0
abb	0	1

(b) Dynamical table

Table 3.2: Observation tables for the language described in Figure 3.4

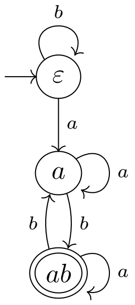


Figure 3.3: Hypothesis for Table 3.2b

A procedure for extending an observation table approximation to make it dynamical was already known to Gold [36]. If we do not have closedness, then there is a row in the lower part of the table that does not occur in the upper part; Gold takes the label of this row and adds it to S, thereby increasing the size of H, denoted $|H|$. (Note that we talk here about the *set* H; it is not necessarily an automaton yet.) If the approximation is not consistent, then there exist $s_1, s_2 \in S$ satisfying $\xi(s_1) = \xi(s_2)$, but there are $a \in A$ and $e \in E$ such that $\xi_\delta(s_1)(a)(e) \neq \xi_\delta(s_2)(a)(e)$; Gold now adds ae to E so that $\xi(s_1)$ and $\xi(s_2)$ become distinguished, and again $|H|$ becomes larger. Thus, after iterating these modifications less than $|M|$ times we must have a dynamical approximation. Note, however, that at this point we may not yet have $|H| = |M|$.

Consider for the alphabet $\{a, b\}$ the language of words that contain an odd number of b's and at least one a, of which the minimal DA is shown in Figure 3.4. Table 3.2a represents the observation structure of $\mathsf{T}(\{\varepsilon, a\}, \{\varepsilon\})$, which is neither closed nor consistent: the row in the lower part labeled by ab does not occur in the upper

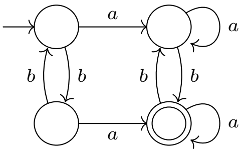


Figure 3.4: Minimal DA accepting words with an odd number of b's and at least one a

part, and although the rows labeled by ε and a are equal, this is not the case for those labeled by b and ab. We fix the closedness defect by adding ab to S. Note that at this point the consistency defect is still present; we fix it by adding b to E. The resulting table is shown in Table 3.2b, and Figure 3.3 shows the corresponding hypothesis. Here a state labeled by a word $s \in S$ actually represents $e(s)$.

3.1.2 Discrimination Trees

So far we have only discussed observation tables as an instance of an approximation in the context of DA. We now turn to a formalism that allows for a more concise representation of a partition of the set of languages.

Fixing a responsiveness or consistency defect in an observation table consists in adding an experiment to E (i.e., a column to the table) in order to distinguish two previously equal rows. To fill the new cells of the resulting tables, a membership query is required for every row in the lower part of the table. This should not be necessary, since we just want a finer classification of those specific equal rows. An alternative for the observation table that allows for such specific refinements is known as the *discrimination tree* [47]. Formally, define the set DT_L of (binary) discrimination trees with leaves in a set L by the following grammar:

$$\mathsf{DT}_L \ ::= \ \mathsf{Leaf}(L) \mid \mathsf{Node}(A^*, \mathsf{DT}_L^2)$$

The function space DT_L^2 could have been replaced by the binary product $\mathsf{DT}_L \times \mathsf{DT}_L$, but the present definition eases a generalization that will be discussed in Section 5.

We will consider *discrimination tree approximations*

$$\mathsf{DT}(S, \tau) = (S, \mathcal{P}S, \sigma_S, \pi_\tau)$$

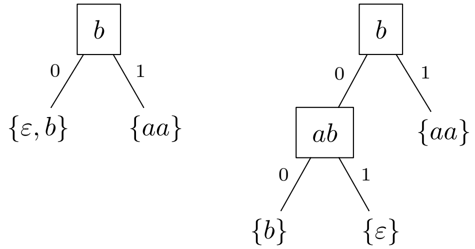
for discrimination trees $\tau \in \mathsf{DT}_{\mathcal{P}S}$, where S is a finite subset of A^* and $\pi_\tau \colon 2^{A^*} \to \mathcal{P}S$ is defined by induction on the structure of the tree τ:

$$\pi_{\mathsf{Leaf}(l)}(f) = l \qquad\qquad \pi_{\mathsf{Node}(u,c)}(f) = \pi_{c(f(u))}(f).$$

An example of such a tree is visualized in Figure 3.5a. We represent inner nodes with boxes, and each edge is labeled by the output that is mapped to its subtree. Concretely, π_τ with τ represented visually classifies a language $f \in 2^{A^*}$ as follows. We begin at the root of the depicted tree. If the current node is a leaf, then the result is the associated label. If the current node is an internal node with label $u \in A^*$, then we use a membership query to find $f(u)$. If $f(u) = 0$, we move to the subtree belonging to the edge with label 0; if $f(u) = 1$, we move to the subtree of the edge with label 1. The process is then repeated with this subtree and the same language. For example, the tree in Figure 3.5b classifies each language that contains the word b into $\{aa\}$. Any language that does not contain b, but does contain ab obtains the label $\{\varepsilon\}$. All other languages are classified into $\{b\}$.

Although the tree acts as a classifier for languages, we can interpret it on words as well. Define the function $\mathsf{sift}_\tau \colon A^* \to \mathcal{P}S$ by $\mathsf{sift}_\tau = \pi_\tau \circ t_{\mathcal{L}}$. *Sifting* a word $u \in A^*$ through the tree consists in starting at the root node and while being at an internal node labeled by $v \in A^*$ observing $\mathcal{L}(uv)$ and proceeding with the associated subtree. For instance, sifting the word bb through the tree in Figure 3.5b yields the label $\{\varepsilon\}$, since the corresponding target language $\mathcal{L}$ does not contain bbb, but does contain $bbab$.

With σ_S being an inclusion, the approximated response can be determined by sifting the words in S through the tree. In fact, this



(a) Example tree **(b)** Consistent tree

Figure 3.5: Discrimination trees for the language given in Figure 3.4; $S = \{\varepsilon, b, aa\}$

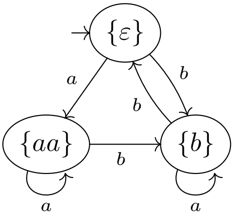


Figure 3.6: Hypothesis for Figure 3.5b

operation allows us to comprehensibly expose the entire observation structure corresponding to $\mathsf{DT}(S, \tau)$.

Proposition 3.17. *For an approximation* $\mathsf{DT}(S, \tau)$*, the observation structure is obtained as*

$$\begin{aligned}
\xi &\colon S \to \mathcal{P}S & \xi(s) &= \mathsf{sift}_\tau(s)\\
\xi_I &\colon 1 \to \mathcal{P}S & \xi_I(*) &= \mathsf{sift}_\tau(\varepsilon)\\
\xi_Y &\colon S \to 2 & \xi_Y(s) &= \mathcal{L}(s)\\
\xi_\delta &\colon S \to (\mathcal{P}S)^A & \xi_\delta(s)(a) &= \mathsf{sift}_\tau(sa).
\end{aligned}$$

Proof. The case for ξ is clear, and ξ_Y is as in Proposition 3.6. Recalling the definition of the input system @ and the fact that $\overline{\mathcal{L}}$ turns @ and Ω into automata such that $t_{\mathcal{L}} \colon @ \to \Omega$ is an automaton homomorphism, we see that

$$\xi_I(*) = (\pi_\tau \circ \overline{\mathcal{L}})(*) = (\pi_\tau \circ t_{\mathcal{L}} \circ \mathsf{init}_@)(*) = (\pi_\tau \circ t_{\mathcal{L}})(\varepsilon) = \mathsf{sift}_\tau(\varepsilon)$$

and, additionally employing the definition and functoriality of $(-)^A$,

$$\begin{aligned}\xi_\delta(s)(a) &= (\pi_\tau^A \circ \delta_\Omega \circ t_{\mathcal{L}} \circ \sigma_S)(s)(a) = (\pi_\tau^A \circ \delta_\Omega \circ t_{\mathcal{L}})(s)(a)\\ &= (\pi_\tau^A \circ t_{\mathcal{L}}^A \circ \delta_@)(s)(a) = ((\pi_\tau \circ t_{\mathcal{L}})^A \circ \delta_@)(s)(a)\\ &= (\pi_\tau \circ t_{\mathcal{L}})^A(\delta_@(s))(a) = (\pi_\tau \circ t_{\mathcal{L}} \circ \delta_@(s))(a)\\ &= \pi_\tau(t_{\mathcal{L}}(\delta_@(s)(a))) = \pi_\tau(t_{\mathcal{L}}(sa)) = \mathsf{sift}_\tau(sa). \qquad \square\end{aligned}$$

Let us now interpret the properties needed for hypothesis constructability in this setting.

Proposition 3.18. *An approximation* $\mathsf{DT}(S, \tau)$ *is initialized if and only if there exists an* $s \in S$ *such that* $\mathsf{sift}_\tau(s) = \mathsf{sift}_\tau(\varepsilon)$*; it is responsive if and only if for all* $s_1, s_2 \in S$ *with* $\mathsf{sift}_\tau(s_1) = \mathsf{sift}_\tau(s_2)$ *we have* $\mathcal{L}(s_1) = \mathcal{L}(s_2)$*.*

Proof. From Proposition 3.7 we know that $\mathsf{DT}(S, \tau)$ is initialized if and only if there exists an $s \in S$ such that $\xi(s) = \xi_I(*)$, and Proposition 3.17 gives us $\xi(s) = \mathsf{sift}_\tau(s)$ and $\xi_I(*) = \mathsf{sift}_\tau(\varepsilon)$ to complete the proof for the first part.

For the second part, Proposition 3.7 gives us that $\mathsf{DT}(S, \tau)$ is consistent if and only if for all $s_1, s_2 \in S$ with $\xi(s_1) = \xi(s_2)$ we have $\xi_Y(s_1) = \xi_Y(s_2)$, so we are done by simply recalling from Proposition 3.17 that for all $s \in S$, $\xi(s) = \mathsf{sift}_\tau(s)$ and $\xi_Y(s) = \mathcal{L}(s)$. □

Corollary 3.19. *A discrimination tree approximation* $\mathsf{DT}(S, \tau)$ *is initialized if* $\varepsilon \in S$*.*

Proposition 3.20. *An approximation* $\mathsf{DT}(S, \tau)$ *is closed if for all* $s \in S$ *and* $a \in A$ *there exists an* $s' \in S$ *such that* $\mathsf{sift}_\tau(s') = \mathsf{sift}_\tau(sa)$*; it is consistent if and only if for all access strings* $s_1, s_2 \in S$ *satisfying* $\mathsf{sift}_\tau(s_1) = \mathsf{sift}_\tau(s_2)$ *we have* $\mathsf{sift}_\tau(s_1 a) = \mathsf{sift}_\tau(s_2 a)$ *for each* $a \in A$*.*

Proof. This follows just by combining Proposition 3.12 with Proposition 3.17. □

The reason for choosing $\mathcal{P}S$ as the set of labels is that we want the leaves of τ to partition the access strings $s \in S$ in such a way that the approximated response can be read directly from the tree. Let us make this restriction formal using terminology from Isberner [42], who defined a similar restriction.

Definition 3.21 (Valid Discrimination Tree)**.** A discrimination tree approximation $\mathsf{DT}(S, \tau)$ is called *valid* if for all $u \in A^*$ and $s \in S$, $s \in \mathsf{sift}_\tau(u)$ if and only if $\mathsf{sift}_\tau(u) = \mathsf{sift}_\tau(s)$.

In practice this property is rather hard to verify exactly, but note that it suffices to have all words in S sift into a leaf they are contained in while ensuring that there is exactly one such leaf. The tree in Figure 3.5a, for instance, is valid. This stronger condition will be the case for all the trees that we maintain, and it allows us to directly read the approximated response from a representation of the tree. In general we do need to determine the other functions of the observation structure using external information.

Valid discrimination tree approximations that do not contain an empty set leaf are always initialized and closed: sifting any word through the tree gives a set of words from S that sift into the same leaf.

Proposition 3.22. *If* $\mathsf{DT}(S, \tau)$ *is valid and for all* $u \in A^*$, $\mathsf{sift}_\tau(u) \neq \emptyset$*, then* $\mathsf{DT}(S, \tau)$ *is initialized and closed.*

Proof. We can show more generally that for each $u \in A^*$ there is an $s \in S$ such that $\mathsf{sift}_\tau(s) = \mathsf{sift}_\tau(u)$: since $\mathsf{sift}_\tau(u)$ is not empty, we pick any $s \in \mathsf{sift}_\tau(u)$, which by validity satisfies $\mathsf{sift}_\tau(s) = \mathsf{sift}_\tau(u)$. □

Proposition 3.23. *If* $\mathsf{DT}(S, \tau)$ *is valid and for each* $u \in A^*$ *we have* $|\mathsf{sift}_\tau(u)| \leq 1$*, then the approximation is responsive and consistent.*

Proof. Suppose there are $s_1, s_2 \in S$ with $\mathsf{sift}_\tau(s_1) = \mathsf{sift}_\tau(s_2)$. By validity and the bound on leaf sizes this common leaf is $\{s_1\} = \{s_2\}$, so trivially $\mathcal{L}(s_1) = \mathcal{L}(s_2)$ and $\xi_\delta(s_1) = \xi_\delta(s_2)$, giving us responsiveness by Proposition 3.18 and consistency by Proposition 3.12. □

The tree in Figure 3.5a is initialized and closed by Proposition 3.22, and it is responsive because $\mathcal{L}(\varepsilon) = \mathcal{L}(b) = 0$. However, it is not consistent: ε and b sift into the same leaf, but this is not the case with a and ba for which we end up in $\{aa\}$ and $\{\varepsilon, b\}$, respectively. To fix this, we can take the experiment that separates the leaves $\{aa\}$ and $\{\varepsilon, b\}$, which is their *lowest common ancestor* b, and split $\mathsf{Leaf}(\{\varepsilon, b\})$ into $\mathsf{Node}(ab, f)$, where $f\colon 2 \to \mathsf{DT}_{\mathcal{P}\{\varepsilon,b,aa\}}$ is given by $f(0) = \mathsf{Leaf}(\{b\})$ and $f(1) = \mathsf{Leaf}(\{\varepsilon\})$. The result is shown in Figure 3.5b. This tree must be responsive and consistent by Proposition 3.23 because all leaves are singletons. Its associated hypothesis is depicted in Figure 3.6.

Let us briefly describe general procedures to extend a valid discrimination tree approximation to make it initialized, responsive, and dynamical. If the valid discrimination tree is not initialized, we simply replace the empty leaf $\mathsf{sift}_{\mathcal{T}}(\varepsilon)$ by $\mathsf{Leaf}(\{\varepsilon\})$ while adding ε to S. If it is not consistent, there exist $s \in S$ and $a \in A$ such that $\mathsf{sift}_{\mathcal{T}}(sa)$ is an empty leaf, which we replace by $\mathsf{Leaf}(\{sa\})$ while adding sa to S. Because these operations add one access string that sifts into its own leaf they must preserve validity. Moreover, these leaves did not exist before, and they end up in the new image H of ξ, which has thus increased.

Regarding the enforcement of responsiveness and consistency, we define the operation of splitting a leaf. This turns a single leaf into a tree with one internal node that distinguishes the words in the original leaf by a given experiment. We formalize it through a function $\mathsf{split}\colon A^* \times \mathcal{P}S \to \mathsf{DT}_{\mathcal{P}S}$ given by

$$\mathsf{split}(v, U) = \mathsf{Node}(v, \lambda y \in 2[\{s \in U \mid \mathcal{L}(sv) = y\}]).$$

Consider in a valid discrimination tree approximation access strings $s_1, s_2 \in S$ such that $\mathsf{sift}_{\mathcal{T}}(s_1) = \mathsf{sift}_{\mathcal{T}}(s_2) = U$. If these exhibit a responsiveness defect—$\mathcal{L}(s_1) \neq \mathcal{L}(s_2)$—we split that leaf into $\mathsf{split}(\varepsilon, U)$. Note that by validity $\{s_1, s_2\} \subseteq U$, so this split yields two distinct subtrees. If, instead, they exhibit a consistency defect—$\mathsf{sift}_{\mathcal{T}}(s_1 a) \neq \mathsf{sift}_{\mathcal{T}}(s_2 a)$ for some $a \in A$—we can find an experiment v

such that $\mathcal{L}(s_1av) \neq \mathcal{L}(s_2av)$ by sifting s_1a and s_2a simultaneously until a discrepancy arises. Our trees are in fact maintained so that v can be described as the label of the lowest common ancestor of the leaves $\mathsf{sift}_\mathcal{T}(s_1a)$ and $\mathsf{sift}_\mathcal{T}(s_2a)$. The common leaf of s_1 and s_2 may now be replaced by $\mathsf{split}(av, U)$, and again there are two distinct subtrees. Because the split function partitions the access strings $s \in U$ by the value of $\mathcal{L}(sv)$ we can easily see that validity is preserved by this operation. Together with the fact that after the splitting there are two distinct leaves as subtrees, we also know that the size of H must increase.

3.2 Learning from Samples

Learning from samples usually refers to passive learning, where a set of words is given along with their value under $\mathcal{L}$ and where no additional queries may be asked. As mentioned in the introduction of this section, we do not consider passive learning algorithms. The samples considered here are of a different nature—they are sets of words that contain a wealth of information about M, which can then be reconstructed almost directly.

Proposition 3.15 was known in a very specific form to Ho [37] and further elaborated on by Arbib and Zeiger [14] and Gold [36], the latter of whom advocates *identification in the limit* that we discuss briefly here. Consider ascending chains of languages $S_1, S_2, \ldots$ and $E_1, E_2, \ldots$ that both in the limit equal A^*. Then for the limit of a natural number k going to infinity, the approximation $\mathsf{T}(S_k, E_k)$ yields a hypothesis that is isomorphic to M.

In the case that M is finite, we even have that the hypothesis of $\mathsf{T}(S_k, E_k)$ for some $k \in \mathbb{N}$ is isomorphic to M, since then there must be a finite set of words that reaches all states of M (e.g. S_k) and a finite set of words that for every pair of states of M contains an experiment to distinguish them (e.g. E_k). To turn this into an algorithm we could assume an upper bound n on the number of states of M and take the complete [36, Theorems 1 and 2] approximation $\mathsf{T}(A^{\leq n}, A^{\leq n})$, where $A^{\leq n}$ contains all words over A of length up to

n. These tables quickly become huge, but it turns out that we can do better: Gold noted that either one of the conditions in Proposition 3.15 can be dropped if we are willing to do some additional work.

Proposition 3.24. *If the approximation observes M and is initialized and closed, then it is complete.*

Proposition 3.25. *If the approximation reaches M and is responsive and consistent, then it is complete.*

Again, these are instances of the more general Theorem 4.11 and Theorem 4.12.

As an example of Proposition 3.24, for each pair of states in the minimal automaton shown in Figure 3.4 there is an experiment in $E = \{a, b, bb\}$ to distinguishes them. By Proposition 3.24 this means that any approximation $\mathsf{T}(S, E)$ that is initialized and closed is complete. We start with $\mathsf{T}(\{\varepsilon\}, E)$, visualized in Table 3.7a. Given $\varepsilon \in S$, the approximation is initialized, and thus it remains to close it, which we have done in Table 3.7b. This approximation is complete—indeed, the set $S = \{\varepsilon, a, b, ab\}$ reaches every state of the minimal automaton in Figure 3.4—so H is isomorphic to M. Note that the approximation was responsive even though no column for the empty word is present. To construct the automaton we did have to inspect this column, but only its upper part (which corresponds to ξ_Y).

We demonstrate the effectiveness of Proposition 3.25 using discrimination trees. An algorithm that (implicitly) uses observation tables was described by Angluin [6], but the use of discrimination trees appears to be new. Assume we are given $S = \{\varepsilon, a, b, ab\}$ that reaches every state in Figure 3.4. Initially, we consider the approximation $\mathsf{DT}(S, \mathsf{Leaf}(S))$. It is consistent, but not responsive: $\mathcal{L}(\varepsilon) = \mathcal{L}(a) = \mathcal{L}(b) = 0$, but $\mathcal{L}(ab) = 1$. Thus, we split into the responsive tree shown in Figure 3.8a. This tree is not consistent: sifting a and aa, we end up in $\{\varepsilon, a, b\}$, whereas ba yields $\{ab\}$. Hence, we split $\{\varepsilon, a, b\}$ with the experiment $a\varepsilon$, where ε is obtained

	a	b	ab
ε	0	0	1
a	0	1	1
b	1	0	1

(a) Initial table

	a	b	ab
ε	0	0	1
a	0	1	1
b	1	0	1
ab	1	0	0
aa	0	1	1
ba	1	0	0
bb	0	0	1
aba	1	0	0

(b) Closed table

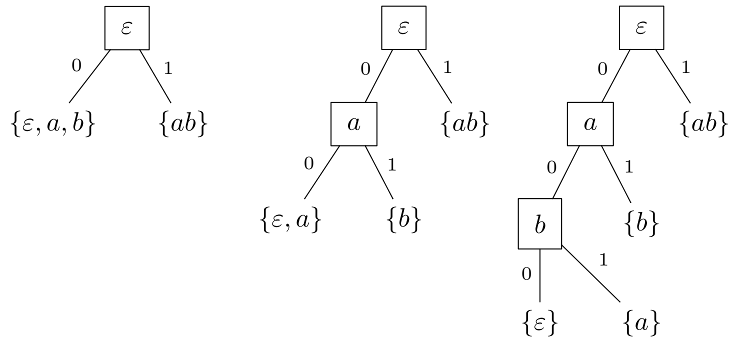
Table 3.7: Observation tables observing the DA in Figure 3.4

as the lowest common ancestor of the aforementioned subsets. Figure 3.8b illustrates the result. Again, the tree is not consistent, but we fix this with another split that produces the tree in Figure 3.8c for which the associated hypothesis is isomorphic to the automaton in Figure 3.4.

3.3 The MAT Model

An algorithm that has received much attention and sparked derivatives that are now being used for practical applications was discovered by Angluin [7]. The main idea here is to assume not a known upper bound on the number of states of M, but a *minimally adequate teacher* who, in addition to membership queries can answer *equivalence queries*, which ask whether a hypothesis is correct, and if not, what a *counterexample* would be. Sufficient information is extracted from these counterexamples to make the resulting algorithm polynomial in the number of states of M, the size of the alphabet A, and the length of the counterexamples.

Thus, starting with a trivial approximation (e.g. $\mathsf{T}(\emptyset, \emptyset)$ or



(a) Responsive tree **(b)** More consistent tree **(c)** Final tree

Figure 3.8: Discrimination trees reaching the DA in Figure 3.4

$\mathsf{DT}(\emptyset, \mathsf{Leaf}(\emptyset)))$, we extend it each time to make it initialized, responsive, and dynamical so as to produce a hypothesis automaton H. Subsequently, the teacher is asked whether H is correct, and unless this is the case we obtain a counterexample.

We explain only the counterexample processing method of Rivest and Schapire [61], which compared to the original method of Angluin reduces the worst case complexity of the number of membership queries that the full algorithm uses. Our discussion at first remains abstract in line with the preceding theory, but the operations performed on the counterexample later on are rather concrete.

We assume that $e\colon S \to H$ has a right inverse: there exists a morphism $i\colon H \to S$ making the diagram below commute.

$$\begin{tikzcd}
H \arrow[d, dashed, "i"'] \arrow[dr, "\mathsf{id}"] & \\
S \arrow[r, "e"] & H
\end{tikzcd}$$

In the DA setting, $i(q)$ for a state of $q \in H$ is a choice of an access string $s \in S$ satisfying $e(s) = q$. Note that i is in general not unique with this property. Furthermore, the possibility of this splitting in the category of sets in general amounts to the axiom of choice, but remember that in practice S is finite.

Using this right inverse, we define a "distorted version" $h\colon @ \to \Omega$ of the total response as the composition below.

$$\begin{tikzcd}
@ \arrow[r, "r"] \arrow[rrrr, bend right, dotted, "h"'] & H \arrow[r, "i"] & S \arrow[r, "\sigma"] & @ \arrow[r, "t_{\mathcal{L}}"] & \Omega
\end{tikzcd}$$

In the DA setting, think of h as assigning to each word $u \in A^*$ not the residual language $t_{\mathcal{L}}(u)$ of u, but the residual language $t_{\mathcal{L}}(s)$ of some $s \in S$ that represents the state of the hypothesis reached by u ($e(s) = r_H(u)$).

In our concrete argument for counterexample analysis, we will be using the following abstract lemmas related to the morphisms i and h.

Lemma 3.26. $\xi_I = \xi \circ i \circ r_H \circ \mathsf{init}_@$.

Proof. Diagram chase using initialization, the definition of ξ_I, and the preservation of initial state maps by r_H:

$$\begin{tikzcd}
I \arrow[r, "\mathsf{init}"] \arrow[rr, bend right, "\mathsf{init}"'] \arrow[rrrdd, bend right, "\xi_I"'] & @ \arrow[r, "r"] & H \arrow[r, "i"] \arrow[rd, "\mathsf{id}"'] & S \arrow[d, "e"'] \arrow[dd, bend left, "\xi"] \\
 & & & H \arrow[d, "m"'] \\
 & & & P
\end{tikzcd}$$

□

Lemma 3.27. $\mathcal{L}_H = \mathsf{out}_\Omega \circ h$.

Proof. Diagram chase:

$$\begin{tikzcd}
@ \arrow[r, "r"] \arrow[drrr, bend left=50, "\mathcal{L}_H"] \arrow[ddd, bend right, "h"'] & H \arrow[d, "i"'] \arrow[dr, "\mathsf{id}"] & & \\
& S \arrow[r, "e"] \arrow[d, "\sigma"'] \arrow[rr, bend right, "\xi_Y"] & H \arrow[r, "\mathsf{out}"] & Y \\
& @ \arrow[d, "t_{\mathcal{L}}"'] \arrow[urr, bend right, "\mathcal{L}"'] & & \\
& \Omega \arrow[uurr, bend right, "\mathsf{out}"'] & &
\end{tikzcd}$$

Note that we use responsiveness and the definition of ξ_Y, as well as homomorphism properties. □

Lemma 3.28. $\xi_\delta \circ i \circ r_H = \mathcal{D}(\xi \circ i \circ r_H) \circ \delta_@$.

Proof. Diagram chase:

$$\begin{tikzcd}
@ \arrow[r, "r"] \arrow[dd, "\delta"'] & H \arrow[r, "i"] \arrow[dr, "\mathsf{id}"'] & S \arrow[d, "e"'] \arrow[dddrr, bend left, "\xi_\delta"] & & \\
& & H \arrow[d, "\delta"'] & & \\
\mathcal{D}@ \arrow[rr, "\mathcal{D}r"] & & \mathcal{D}H \arrow[d, "\mathcal{D}i"'] \arrow[dr, "\mathcal{D}\mathsf{id}"] & & \\
& & \mathcal{D}S \arrow[r, "\mathcal{D}e"] \arrow[rr, bend right, "\mathcal{D}\xi"'] & \mathcal{D}H \arrow[r, "\mathcal{D}m"] & \mathcal{D}P
\end{tikzcd}$$

On the right we use dynamism and at the bottom the factorization of ξ. The region on the left asserts that r_H is a dynamorphism. □

From now on we restrict our attention to the DA setting. We assume that the hypothesis can be constructed and that a counterexample has been provided by the teacher. A counterexample is a word $z \in A^*$ satisfying $\mathcal{L}(z) \neq \mathcal{L}_H(z)$. The idea is to extract from this counterexample a new experiment that either directly increases the size of H or causes a defect in the initialization or closedness of the approximation so that using the procedures defined earlier for specific approximations to fix this defect H will also become larger.

More specifically, consider first initialization as obtained in Proposition 3.7: there exists an $s \in S$ such that $\xi(s) = \xi_I(*)$. With the definitions of ξ and ξ_I this becomes

$$(\pi \circ t_{\mathcal{L}} \circ \sigma)(s) = (\pi \circ \overline{\mathcal{L}})(*).$$

We are interested in finding an experiment $v \in A^*$ that breaks this equality as follows:[2]

$$(t_{\mathcal{L}} \circ \sigma)(s)(v) \neq \overline{\mathcal{L}}(*)(v).$$

Incorporating such v into π will either distinguish $\xi(s)$ from some $\xi(s')$ with $s' \in S$ such that $\xi(s') = \xi_I(*)$, or it will harm the initialization of the approximation, since $\xi(s) \neq \xi_I(*)$ after this modification. In the end, the size of H must increase.

Incorporating v into π for the approximations that we have seen works as expected. For an observation table $\mathsf{T}(S, E)$ we simply add v to E, and for a discrimination tree $\mathsf{DT}(S, \tau)$ we replace $\mathsf{sift}_\tau(s)$ by $\mathsf{split}(v, \mathsf{sift}_\tau(s))$.

In the proposition below that provides such v we add a condition that is equivalent to $\mathcal{L}(z) \neq \mathcal{L}(i(r_H(\varepsilon)) \cdot z)$. We will in the end make a case distinction on this condition to decide whether to apply the method described above regarding an initialization defect or to aim for a closedness defect.

Proposition 3.29. *If z is a counterexample for the hypothesis H and $\mathcal{L}(z) \neq (h \circ \mathsf{init}_@)(*)(z)$, then there are $s \in S$ and $v \in A^*$ such that $\xi(s) = \xi_I(*)$, but $(t_{\mathcal{L}} \circ \sigma)(s)(v) \neq \overline{\mathcal{L}}(*)(v)$.*

[2] The word $v \in A^*$ is actually better seen as an evaluation function $\mathsf{ev}_v \colon 2^{A^*} \to 2$ defined by $\mathsf{ev}_v(f) = f(v)$. The inequality that follows can then be expressed as $(\mathsf{ev}_v \circ t_{\mathcal{L}} \circ \sigma)(s) \neq (\mathsf{ev}_v \circ \overline{\mathcal{L}})(*)$. What is really happening here is that we are trying to combine two approximations. Note however, that we do not want the full power of ev_v; we only use it to distinguish two specific classes that are at the moment identified by π.

Proof. Define $s = (i \circ r_H \circ \mathsf{init}_@)(*)$ and $v = z$. Using Lemma 3.26, we have $\xi(s) = (\xi \circ i \circ r_H \circ \mathsf{init}_@)(*) = \xi_I(*)$. Furthermore, the inequality comes down to our assumption:

$$\begin{aligned}(t_{\mathcal{L}} \circ \sigma)(s)(z) = (t_{\mathcal{L}} \circ \sigma \circ i \circ r_H \circ \mathsf{init}_@)(*)(z) = (h \circ \mathsf{init}_@)(*)(z) \\ \neq \mathcal{L}(z) = \overline{\mathcal{L}}(*)(z). \ \square\end{aligned}$$

Now recall from Proposition 3.12 the definition of a witness for closedness: given $s \in S$ and $a \in A$ there exists an $s' \in S$ such that $\xi(s') = \xi_\delta(s)(a)$. Again, we rewrite this using definitions of the observation structure as

$$(\pi \circ t_{\mathcal{L}} \circ \sigma)(s') = (\pi^A \circ \delta_\Omega \circ t_{\mathcal{L}} \circ \sigma)(s)(a),$$

and this time we seek a word $v \in A^*$ such that

$$(t_{\mathcal{L}} \circ \sigma)(s')(v) \neq (\delta_\Omega \circ t_{\mathcal{L}} \circ \sigma)(s)(a)(v).$$

We can then incorporate v into π analogous to the previous case, and the size of H will increase for similar reasons.

Proposition 3.30. *If z is a counterexample for the hypothesis H and $\mathcal{L}(z) = (h \circ \mathsf{init}_@)(*)(z)$, then there are $s, s' \in S$, $a \in A$, and $v \in A^*$ such that $\xi(s') = \xi_\delta(s)(a)$, but*

$$(t_{\mathcal{L}} \circ \sigma)(s')(v) \neq (\delta_\Omega \circ t_{\mathcal{L}} \circ \sigma)(s)(a)(v).$$

Proof. We prove first that

$$h(\varepsilon)(z) \neq h(z)(\varepsilon). \tag{3.1}$$

Observe that

$$\begin{aligned}
h(\varepsilon)(z) &= (h \circ \mathsf{init}_@)(*)(z) && (\varepsilon = \mathsf{init}_@(*)) \\
&= \mathcal{L}(z) && \text{(assumption)} \\
&\neq \mathcal{L}_H(z) && \text{(counterexample)} \\
&= (\mathsf{out}_\Omega \circ h)(z) && \text{(Lemma 3.27)} \\
&= h(z)(\varepsilon) && \text{(definition of } \mathsf{out}_\Omega\text{)}.
\end{aligned}$$

Imagine gradually transitioning from the left expression in (3.1) to the right one. This leads us to realize that there must be a breakpoint—i.e., there are $u, v \in A^*$ and $a \in A$ such that

$$h(ua)(v) \neq h(u)(av),$$

which may be expressed as a failure of h to be a dynamorphism:

$$(h^A \circ \delta_@)(u)(a)(v) \neq (\delta_\Omega \circ h)(u)(a)(v). \tag{3.2}$$

Define $s = (i \circ r_H)(u)$ and $s' = (i \circ r_H \circ \delta_@(u))(a)$. These give us the required inequality

$$\begin{aligned}
&(t_\mathcal{L} \circ \sigma)(s')(v) \\
&= (t_\mathcal{L} \circ \sigma \circ i \circ r_H \circ \delta_@(u))(a)(v) && \text{(definition of } s'\text{)} \\
&= (h \circ \delta_@(u))(a)(v) && \text{(definition of } h\text{)} \\
&= (h^A \circ \delta_@)(u)(a)(v) && \text{(definition of } (-)^A\text{)} \\
&\neq (\delta_\Omega \circ h)(u)(a)(v) && (3.2) \\
&= (\delta_\Omega \circ t_\mathcal{L} \circ \sigma \circ i \circ r_H)(u)(a)(v) && \text{(definition of } h\text{)} \\
&= (\delta_\Omega \circ t_\mathcal{L} \circ \sigma)(s)(a)(v) && \text{(definition of } s\text{).}
\end{aligned}$$

It remains to show that $\xi(s') = \xi_\delta(s)(a)$. Rewriting these as

$$\begin{aligned}
\xi(s') &= (\xi \circ i \circ r_H \circ \delta_@(u))(a) = ((\xi \circ i \circ r_H)^A \circ \delta_@)(u)(a) \\
\xi_\delta(s)(a) &= (\xi_\delta \circ i \circ r_H)(u)(a),
\end{aligned}$$

we see that we can conclude by applying Lemma 3.28. □

Thus, we query $\mathcal{L}(z)$ and

$$\begin{aligned}
&(h \circ \mathsf{init}_@)(*)(z) \\
&= (t_\mathcal{L} \circ \sigma \circ i \circ r_H \circ \mathsf{init}_@)(*)(z) && \text{(definition of } h\text{)} \\
&= (t_\mathcal{L} \circ \sigma \circ i \circ r_H)(\varepsilon)(z) && \text{(definition of } \mathsf{init}_@\text{)} \\
&= t_\mathcal{L}((\sigma \circ i \circ r_H)(\varepsilon))(z) \\
&= \mathcal{L}(i(r_H(\varepsilon)) \cdot z)
\end{aligned}$$

to decide whether to apply Proposition 3.29 or Proposition 3.30. If $\varepsilon \in S$, we can choose $i(r_H(\varepsilon)) = \varepsilon$, and then we always go directly for the latter.

In practice, the breakpoint referred to in the above proof is found efficiently by means of a binary search (see the exposition of Isberner and Steffen [43] for alternatives). The learning algorithm then has a membership query complexity of $\mathcal{O}(kn^2 + n \log m)$, where $k = |A|$, $n = |M|$, and m is the length of the longest counterexample that was given. This is regardless of whether an observation table [61] or a discrimination tree [47, 18] approach is used. Because the size of H will increase after each counterexample, there can never be more than n equivalence queries.

Recall the hypothesis from Figure 3.3 and note that the labeling of the states represents a choice for the function i (which in this case is unique). The hypothesis is not correct, for

$$\mathcal{L}_H(ba) = 0 \neq 1 = \mathcal{L}(ba).$$

Suppose the teacher provides us with this counterexample ba. Since $i(\mathsf{init}_H(*)) = \varepsilon$, we proceed to applying Proposition 3.30. To calculate $h(x)(y)$ we find the state that H ends up in after reading x, read its state label s, and query $\mathcal{L}(sy)$. Thus,

$$\begin{aligned} h(ba)(\varepsilon) &= \mathcal{L}(a) &&= 0 \\ h(b)(a) &= \mathcal{L}(a) &&= 0 \\ h(\varepsilon)(ba) &= \mathcal{L}(ba) &&= 1, \end{aligned}$$

which means that the experiment a is to be added to Table 3.2b.

3.4 Discussion

Balcázar et al. [18] made a first attempt at unifying the existing active automata learning algorithms for DA. More recently, an extensively detailed algorithmic account of active automata learning has been given by Isberner [42]. We do not at all intend to replace these frameworks; instead, they can be regarded as complementary to ours. Many of their results are of a practical nature, whereas we focus on more conceptual results that provide simple insights on an abstract level. The discussed examples are meant for illustration and to provide evidence for the expressiveness of our framework.

For similar reasons, our results are compatible with the domain-specific optimizations of Hungar et al. [41], which are just assumptions regarding the language $\mathcal{L}$ that allow to reduce the number of membership queries. Furthermore, we note that the TTT algorithm [44], which is the most efficient known active automata learning algorithm, in our framework uses an approximation that is just a discrimination tree. However, because at our abstract level we do not try to enforce any specific algorithms, its use of additional administration and optimization subroutines is not prohibited.

In the work of Isberner [42], the function $\pi\colon 2^{A^*} \to P$ of the approximation is replaced by a *black box classifier* $\kappa\colon A^* \to P$, which is called *valid* if for all $u_1, u_2 \in A^*$ that satisfy $t_{\mathcal{L}}(u_1) = t_{\mathcal{L}}(u_2)$ we have $\kappa(u_1) = \kappa(u_2)$. Indeed, any of our functions π induces a valid black box classifier $\kappa = \pi \circ t_{\mathcal{L}}$: if $t_{\mathcal{L}}(u_1) = t_{\mathcal{L}}(u_2)$, then certainly $\pi(t_{\mathcal{L}}(u_1)) = \pi(t_{\mathcal{L}}(u_2))$. It is to be expected that this conversion respects the construction of the hypothesis so that Isberner's results apply also to approximations in the setting of DA that may be studied in our framework. A formal confirmation of this expectation is left as future work.

We will show in Section 4, where we generalize our concept of an approximation, that without additional assumptions valid black box classifiers can also be characterized abstractly. The above relation already suggests that they are more general than our approximations, which carries over to the abstract definition. However, the elegant symmetry and conceptual simplicity of approximations is sacrificed, which is why the present section does not use black box classifiers. On the other hand, our discussion of discrimination trees would have been simplified by such a change: we defined (the valid black box abstraction) $\mathsf{sift}_{\mathcal{T}}$ based on $\pi_{\mathcal{T}}$, the latter of which subsequently becomes essentially redundant.

4 Unifying Learning and Minimization

Active automata learning as we have seen here has been developed based on automaton minimization theory, but so far we have not related automata learning algorithms to minimization algorithms. In order to facilitate this, we generalize the concept of an approximation by basing it merely on an automaton.

Definition 4.1 (Automaton Wrapper). A *wrapper* for an automaton Q is a tuple (S, P, α, β) comprising objects S and P with morphisms $\alpha\colon S \to Q$ and $\beta\colon Q \to P$ in $\mathbf{B}$.

Automaton wrappers subsume approximations in the following manner: an approximation (S, P, σ, π) induces an M-wrapper $(S, P, r_M \circ \sigma, \pi \circ o_M)$ for the minimal realization M of the language under consideration in learning. We redefine the observation structure for the new concept.

Definition 4.2 (Observation Structure). The *observation structure* associated with a Q-wrapper (S, P, α, β) is given by the following morphisms:

$$\begin{array}{lclcl}
\xi & : & S \to P & : & \beta \circ \alpha \\
\xi_I & : & I \to P & : & \beta \circ \mathsf{init}_Q \\
\xi_Y & : & S \to Y & : & \mathsf{out}_Q \circ \alpha \\
\xi_\delta & : & S \to \mathcal{D}P & : & \mathcal{D}\beta \circ \delta_Q \circ \alpha.
\end{array}$$

For approximations, this is equivalent to the old Definition 3.14. We show this fact slightly more generally: if a Q-wrapper factors through automaton homomorphisms surrounding Q, then we can rewrite the observation structure independent of Q, in terms of the surrounding automata. The practical relevance of this is that these automata may be known, whereas Q is not. In the learning context this was the case for @ and Ω, which were extended to automata using a language.

Proposition 4.3. *Let Q, U, and V be automata with automaton homomorphisms $u\colon U \to Q$ and $v\colon Q \to V$. The observation structure associated with a Q-wrapper $(S, P, u \circ \sigma, \pi \circ v)$ for morphisms $\sigma\colon S \to U$ and $\pi\colon V \to P$ in $\mathbf{B}$ is obtained as follows:*

$$\begin{array}{lclcl}
\xi & : & S \to P & : & \pi \circ v \circ u \circ \sigma \\
\xi_I & : & I \to P & : & \pi \circ \mathsf{init}_V \\
\xi_Y & : & S \to Y & : & \mathsf{out}_U \circ \sigma \\
\xi_\delta & : & S \to \mathcal{D}P & : & \mathcal{D}\pi \circ \delta_V \circ v \circ u \circ \sigma.
\end{array}$$

Proof. For ξ this is immediate from its definition. We derive the other forms from homomorphism properties:

$$\begin{tikzcd}
I \arrow[d, "\mathsf{init}"] \arrow[dd, bend right, "\mathsf{init}"'] \arrow[ddr, bend left, "\xi_I"] & \\
Q \arrow[d, "v"] & \\
V \arrow[r, "\pi"] & P
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, "\sigma"] \arrow[ddr, bend right, "\xi_Y"'] & U \arrow[d, "u"'] \arrow[dd, bend left, "\mathsf{out}"] \\
& Q \arrow[d, "\mathsf{out}"'] \\
& Y
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[d, "\sigma"'] \arrow[ddddrr, bend left, "\xi_\delta"] & & \\
U \arrow[d, "u"'] & & \\
Q \arrow[r, "\delta"] \arrow[d, "v"'] & \mathcal{D}Q \arrow[d, "\mathcal{D}v"'] & \\
V \arrow[r, "\delta"] & \mathcal{D}V \arrow[r, "\mathcal{D}\pi"] & \mathcal{D}P
\end{tikzcd}$$

□

As promised in Section 3.4, we show at this point how valid black box classifiers [42] can be characterized abstractly.

Definition 4.4 (Black Box Classifier)**.** A pair (P, κ) of an object P and a morphism $\kappa\colon @ \to P$ in $\mathbf{B}$ is called a *black box classifier*, which is *valid* provided that there exists a morphism $x\colon M \to P$ making the diagram below commute.

$$\begin{tikzcd}
@ \arrow[r, two heads, "r_M"] \arrow[dr, "\kappa"'] & M \arrow[d, dotted, "x"] \\
& P
\end{tikzcd}$$

Proposition 4.5. *In the DA setting, a black box classifier (P, κ) is valid if and only if for all $u_1, u_2 \in A^*$ that satisfy $t_{\mathcal{L}}(u_1) = t_{\mathcal{L}}(u_2)$ we have $\kappa(u_1) = \kappa(u_2)$.*

Proof. Because o_M is injective and $o_M \circ r_M = t_{\mathcal{L}}$ it follows that $\mathsf{ker}(r_M) = \mathsf{ker}(t_{\mathcal{L}})$, so we may conclude with Lemma 2.17. □

We thus become interested in M-wrappers $(S, P, r_M \circ \sigma, x)$ for morphisms $\sigma \colon S \to @$. As might be expected, x is not needed explicitly in the associated observation structure.

Proposition 4.6. *For a valid black box classifier (P, κ) with $x \colon M \to P$ as in Definition 4.4, the observation structure associated with an M-wrapper $(S, P, r_M \circ \sigma, x)$, where $\sigma \colon S \to @$, is obtained as follows:*

$$\begin{array}{lclcl}
\xi & : & S \to P & : & \kappa \circ \sigma \\
\xi_I & : & I \to P & : & \kappa \circ \mathsf{init}_@ \\
\xi_Y & : & S \to Y & : & \mathcal{L} \circ \sigma \\
\xi_\delta & : & S \to \mathcal{D}P & : & \mathcal{D}\kappa \circ \delta_@ \circ \sigma.
\end{array}$$

Proof. The case for ξ_Y can be taken from Proposition 4.3 (recalling that $\mathsf{out}_@ = \mathcal{L}$), and regarding ξ and ξ_I we use some basic equalities and the fact that r_M is an input system homomorphism:

$$\begin{tikzcd}
S \arrow[r, "\sigma"] \arrow[ddr, bend right, "\xi"'] & @ \arrow[d, "r"'] \arrow[dd, bend left, "\kappa"] \\
& M \arrow[d, "x"'] \\
& P
\end{tikzcd}
\qquad
\begin{tikzcd}
I \arrow[r, "\mathsf{init}"] \arrow[dr, "\mathsf{init}"'] \arrow[ddr, bend right, "\xi_I"'] & @ \arrow[d, "r"'] \arrow[dd, bend left, "\kappa"] \\
& M \arrow[d, "x"'] \\
& P
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, "\sigma"] \arrow[dddrr, bend right, "\xi_\delta"'] & @ \arrow[d, "r"] \arrow[dr, "\delta"] & \\
& M \arrow[dr, "\delta"'] & \mathcal{D}@ \arrow[d, "\mathcal{D}r"'] \arrow[dd, bend left, "\mathcal{D}\kappa"] \\
& & \mathcal{D}M \arrow[d, "\mathcal{D}x"'] \\
& & \mathcal{D}P
\end{tikzcd}$$

□

We should note again that this kind of a wrapper is more general than our approximation, so in some situations it might be necessary to use this concept instead.

For most of this section we fix an arbitrary Q-wrapper (S, P, α, β) for some automaton Q. We simply copy the definitions of initialization, responsiveness, dynamism, closedness, and consistency as we defined them for approximations to the new setting with wrappers. These depended only on the observation structure. Note that Proposition 3.11 still holds and that the hypothesis is also still a valid concept. Furthermore, the following definition is clearly in harmony with Definition 3.3.

Definition 4.7 (Complete Wrapper)**.** We say that the Q-wrapper *reaches* Q if $\alpha \in \mathcal{E}$; it *observes* Q if $\beta \in \mathcal{M}$. The wrapper is called *complete* if it both reaches and observes Q.

A trivial but satisfying example of a complete wrapper is the H-wrapper (S, P, e, m) whenever the hypothesis can be constructed.

We now set out to prove more abstract versions of the results mentioned in Section 3.2 that fueled the learning algorithms. If the wrapper reaches Q, there exists a diagonal ϕ making the diagram below on the left commute; if the wrapper observes Q, there exists a diagonal ψ making the diagram on the right commute.

$$\begin{tikzcd}
S \arrow[r, two heads, "\alpha"] \arrow[d, "e"'] & Q \arrow[d, "\beta"] \arrow[dl, dotted, "\phi"'] \\
H \arrow[r, tail, "m"] & P
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
S \arrow[r, two heads, "e"] \arrow[d, "\alpha"'] & H \arrow[d, "m"] \arrow[dl, dotted, "\psi"'] \\
Q \arrow[r, tail, "\beta"] & P
\end{tikzcd}$$

If the wrapper is complete, ϕ and ψ are inverse to each other.

Lemma 4.8. *A wrapper that reaches Q is initialized and closed; a wrapper that observes Q is responsive and consistent.*

Proof. Assume the wrapper reaches Q, and define init_H and close as the compositions indicated in the diagrams below, which also show that these give us initialization and closedness.

$$\begin{tikzcd}
I \arrow[r, "\mathsf{init}_Q"'] \arrow[rr, bend left, "\xi_I"] \arrow[dr, dotted, "\mathsf{init}_H"'] & Q \arrow[r, "\beta"'] \arrow[d, "\phi"] & P \\
 & H \arrow[ur, "m"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, "\alpha"'] \arrow[rrr, dotted, bend left, "\mathsf{close}"] \arrow[drrr, bend right, "\xi_\delta"'] & Q \arrow[r, "\delta_Q"'] & \mathcal{D}Q \arrow[r, "\mathcal{D}\phi"'] \arrow[dr, "\mathcal{D}\beta"'] & \mathcal{D}H \arrow[d, "\mathcal{D}m"] \\
 & & & \mathcal{D}P
\end{tikzcd}$$

Now assume instead that the wrapper observes Q, and define out_H and cons as below, where additionally we indicate responsiveness and consistency.

$$\begin{tikzcd}
 & H \arrow[d, "\psi"] \arrow[dr, dotted, "\mathsf{out}_H"] & \\
S \arrow[ur, "e"] \arrow[r, "\alpha"] \arrow[rr, bend right, "\xi_Y"'] & Q \arrow[r, "\mathsf{out}_Q"] & Y
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[d, "e"'] \arrow[dr, "\alpha"] \arrow[rrrd, bend left, "\xi_\delta"] & & & \\
H \arrow[r, "\psi"] \arrow[rrr, dotted, bend right, "\mathsf{cons}"'] & Q \arrow[r, "\delta_Q"] & \mathcal{D}Q \arrow[r, "\mathcal{D}\beta"] & \mathcal{D}H
\end{tikzcd}$$

□

Lemma 4.9. *Assume the hypothesis can be constructed. If the wrapper reaches Q, then ϕ is an automaton homomorphism; if the wrapper observes Q, then ψ is an automaton homomorphism.*

Proof. Assume first that the wrapper reaches Q. We will show that $\phi \circ \mathsf{init}_Q = \mathsf{init}_H$, $\mathsf{out}_H \circ \phi = \mathsf{out}_Q$ and $\delta_H \circ \phi = \mathcal{D}\phi \circ \delta_Q$ by precomposing with the epi α and composing with the monos m and $\mathcal{D}m$. Note that these proofs use initialization, responsiveness, and dynamism.

$$\begin{tikzcd}
I \arrow[r, "\mathsf{init}_H"] \arrow[dd, "\mathsf{init}_Q"'] \arrow[dr, "\xi_I"] & H \arrow[d, tail, "m"] \\
 & P \\
Q \arrow[ur, "\beta"] \arrow[r, "\phi"'] & H \arrow[u, tail, "m"']
\end{tikzcd}
\qquad
\begin{tikzcd}
Q \arrow[r, "\phi"] & H \arrow[dd, "\mathsf{out}_H"] \\
S \arrow[u, two heads, "\alpha"] \arrow[d, two heads, "\alpha"'] \arrow[ur, "e"] \arrow[dr, "\xi_Y"] & \\
Q \arrow[r, "\mathsf{out}_Q"'] & Y
\end{tikzcd}
\qquad
\begin{tikzcd}
Q \arrow[d, "\delta_Q"'] & S \arrow[l, two heads, "\alpha"'] \arrow[r, two heads, "\alpha"] \arrow[dr, "e"] \arrow[dd, "\xi_\delta"] & Q \arrow[d, "\phi"] \\
\mathcal{D}Q \arrow[d, "\mathcal{D}\phi"'] \arrow[dr, "\mathcal{D}\beta"] & & H \arrow[d, "\delta_H"] \\
\mathcal{D}H \arrow[r, tail, "\mathcal{D}m"] & \mathcal{D}P & \mathcal{D}H \arrow[l, tail, "\mathcal{D}m"']
\end{tikzcd}$$

Now assume instead that the wrapper observes Q. We show $\psi \circ \mathsf{init}_H = \mathsf{init}_Q$, $\mathsf{out}_Q \circ \psi = \mathsf{out}_H$, and $\mathcal{D}\psi \circ \delta_H = \delta_Q \circ \psi$ by precomposing with the epi e and composing with the monos β and

$\mathcal{D}\beta$. The proof here is quite similar to the previous one.

$$\begin{tikzcd}
I \arrow[r, "\mathsf{init}_Q"] \arrow[dd, "\mathsf{init}_H"'] \arrow[dr, "\xi_I"'] & Q \arrow[d, tail, "\beta"] \\
 & P \\
H \arrow[ur, "m"] \arrow[r, "\psi"'] & Q \arrow[u, tail, "\beta"']
\end{tikzcd}
\qquad
\begin{tikzcd}
H \arrow[r, "\psi"] & Q \arrow[dd, "\mathsf{out}_Q"] \\
S \arrow[u, two heads, "e"] \arrow[d, two heads, "e"'] \arrow[ur, "\alpha"'] \arrow[dr, "\xi_Y"] & \\
H \arrow[r, "\mathsf{out}_H"'] & Y
\end{tikzcd}
\qquad
\begin{tikzcd}
H \arrow[d, "\delta_H"'] & S \arrow[l, two heads, "e"'] \arrow[r, two heads, "e"] \arrow[dr, "\alpha"] \arrow[dd, "\xi_\delta"] & H \arrow[d, "\psi"] \\
\mathcal{D}H \arrow[d, "\mathcal{D}\psi"'] \arrow[dr, "\mathcal{D}m"] & & Q \arrow[d, "\delta_Q"] \\
\mathcal{D}Q \arrow[r, tail, "\mathcal{D}\beta"] & \mathcal{D}P & \mathcal{D}Q \arrow[l, tail, "\mathcal{D}\beta"']
\end{tikzcd}$$

□

Theorem 4.10. *If the Q-wrapper is complete, then the hypothesis is isomorphic to Q.*

Proof. We have already seen that the hypothesis can indeed be constructed (Lemma 4.8). In addition, both ϕ and ψ exist and are inverse to each other, so with Lemma 4.9 we know that these are actually automaton isomorphisms. □

Theorem 4.11. *If Q is reachable and the wrapper observes Q and is initialized and closed, then it is complete.*

Proof. Assume Q is reachable and the wrapper observes Q and is initialized and closed. Lemma 4.8 tells us that the hypothesis can be constructed, and from Lemma 4.9 we know that ψ is an input system homomorphism. By initiality of @ thus $\psi \circ r_H = r_Q$, and then ψ must be an epi by (F3) because r_Q is epic. Since ψ is defined to satisfy $\alpha = \psi \circ e$ and compositions of morphisms in $\mathcal{E}$ are also in $\mathcal{E}$ (F1), the wrapper reaches Q. □

Theorem 4.12. *If Q is observable and the wrapper reaches Q and is responsive and consistent, then it is complete.*

Proof. Assume Q is observable and the wrapper reaches Q and is responsive and consistent. Lemma 4.8 shows the hypothesis can be constructed, and by Lemma 4.9 ϕ is an output system homomorphism. Then from finality of Ω it follows that $o_H \circ \phi = o_Q$, which is a mono, and so ϕ must be monic by (F3). Using that $\beta = m \circ \phi$, we conclude with (F1) that the wrapper observes Q. □

Let us illustrate the relevance of these last two theorems in a minimization context by contemplating different ways to minimize an automaton Q. That is, we seek a factorization of $t_{\mathcal{L}_Q}$ in **Aut**. Of course, conceptually, such a factorization can be obtained directly:

$$\begin{tikzcd} @ \arrow[r, two heads, "r_M"] \arrow[rr, bend right, "t_{\mathcal{L}_Q}"'] & M \arrow[r, tail, "o_M"] & \Omega \end{tikzcd}$$

However, this does not directly suggest any practical algorithm.

The usual method to minimize an automaton works sequentially: we first take the reachable part of the automaton and subsequently make it observable. That is, we factorize r_Q to obtain an automaton R, and then we factorize o_R to find M (up to isomorphism):

$$\begin{tikzcd} @ \arrow[r, two heads, "r_R"] \arrow[rr, bend right, "r_Q"'] & R \arrow[r, tail, "h_{R,Q}"] & Q \end{tikzcd} \qquad \begin{tikzcd} R \arrow[r, two heads, "h_{R,M}"] \arrow[rr, bend right, "o_R"'] & M \arrow[r, tail, "o_M"] & \Omega \end{tikzcd}$$

Note that by initiality $r_M = h_{R,M} \circ r_R$, which is in $\mathcal{E}$, so that M here is indeed minimal. We can also express this method in terms of automaton wrappers.

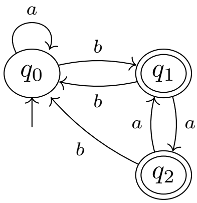
Proposition 4.13 (Sequential Minimization)**.** *If for $\alpha\colon S \to R$ and $\beta\colon M \to P$, $W_R = (S, Q, \alpha, h_{R,Q})$ is an initialized and closed R-wrapper and $W_M = (S, P, h_{R,M} \circ \alpha, \beta)$ is a responsive and consistent M-wrapper, then W_M is complete.*

Proof. Since R is reachable, W_R is complete by Theorem 4.11. In particular, $\alpha \in \mathcal{E}$. Then W_M reaches M because $h_{R,M} \in \mathcal{E}$ by definition and $\mathcal{E}$ is closed under composition. As M is observable, we read from Theorem 4.12 that W_M is complete. □

We take as an example the automaton from Figure 2.3, which we call Q. Starting with the initialized wrapper $(\{\varepsilon\}, Q, r_R \circ \sigma_{\{\varepsilon\}}, h_{R,Q})$ for R, we obtain in the expected way its closed extension, which is shown in Figure 4.1a. Instead of taking this to be W_R, we construct from it the reachable automaton R shown in Figure 4.1b and

ε	q_0
b	q_1
ba	q_2
a	q_0
bb	q_0
baa	q_1
bab	q_0

(a)



(b)

Figure 4.1: Reachability table for Figure 2.3 and its reachable hypothesis R

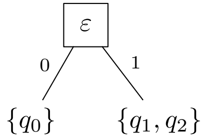


Figure 4.2: Splitting tree for R

take $W_R = (R, Q, \mathsf{id}_R, h_{R,Q})$. Although this is unnecessary work, it demonstrates the relation to a well-known minimization algorithm more precisely. We now move to an M-wrapper $(R, 2^R, h_{R,M}, \pi_\tau)$ for $\tau = \mathsf{Leaf}(R) \in \mathsf{DT}_{2^R}$, which works very much like a normal discrimination tree, called a *splitting tree* in a minimization context [49]. It is not responsive, so we split into the tree shown in Figure 4.2. At this point we also have consistency, which implies that the corresponding hypothesis is isomorphic to M and the automaton in Figure 2.2.

Note that the order of this sequential minimization can be reversed: we could first identify the observable equivalent O of Q by factorizing o_Q and then factorize r_O to obtain M.

$$\begin{tikzcd}
Q \arrow[r, two heads, "h_{Q,O}"] \arrow[rr, bend right, "o_Q"'] & O \arrow[r, tail, "o_O"] & \Omega
\end{tikzcd}
\qquad
\begin{tikzcd}
@ \arrow[r, two heads, "r_M"] \arrow[rr, bend right, "r_O"'] & M \arrow[r, tail, "h_{M,O}"] & O
\end{tikzcd}$$

Proposition 4.14 (Alternative Sequential Minimization). *If for $\alpha\colon S \to M$ and $\beta\colon O \to P$, $W_O = (Q, P, h_{Q,O}, \beta)$ is a responsive and consistent O-wrapper and $W_M = (S, P, \alpha, \beta \circ h_{M,O})$ is an initialized and closed M-wrapper, then W_M is complete.*

	ε	b
q_0	0	1
q_1	1	0
q_2	1	0
q_3	0	0

(a) Observability table

	ε	b
ε	0	1
b	1	0
a	0	1
ba	1	0
bb	0	1

(b) Final table

Table 4.3: Tables for the alternative sequential method applied to Figure 2.3

	ε	b
ε	0	1
b	1	0
ba	1	0
a	0	1
bb	0	1
baa	1	0
bab	0	1

Table 4.4: Figure 4.1a composed with Table 4.3a

Proof. Since O is observable, W_O is complete by Theorem 4.12. Then $\beta \in \mathcal{M}$ and W_M observes M because $h_{M,O} \in \mathcal{M}$ by definition and $\mathcal{M}$ is closed under composition. As M is reachable, we read from Theorem 4.11 that W_M is complete. □

Again we give an example with Q from Figure 2.3, and now we stick close to Proposition 4.14. In Table 4.3a is shown an observation table with rows labeled by states instead of access strings. The table was initialized by creating a row for each state and initially taking only a column with the empty word. Note that the lower part has been left out, for we would label for each state q and input symbol a a row in the lower part by the state $\delta_Q(q)(a)$, which must already be in the upper part. A cell for a row q and column e is filled by determining if e is in the language accepted by q. Table 4.3a has been made consistent in the expected way. If we were to create the corresponding hypothesis, it would be isomorphic to the observable DA O. Instead, we move to a new table—this time an ordinary observation table—by initializing $S = \{\varepsilon\}$ while keeping the column labels of the old table. Closing this table yields Table 4.3b with an associated hypothesis isomorphic to M.

A different approach is to factorize both r_Q and o_Q independently to obtain both R and O. We can then factorize the homomorphism from R to O to obtain M.

$$\begin{tikzcd}
@ \arrow[r, two heads, "r_R"] \arrow[rr, bend right, "r_Q"'] & R \arrow[r, tail, "h_{R,Q}"] & Q \arrow[r, two heads, "h_{Q,O}"] \arrow[rr, bend right, "o_Q"'] & O \arrow[r, tail, "o_O"] & \Omega
\end{tikzcd}
\qquad
\begin{tikzcd}
R \arrow[r, two heads, "h_{R,M}"] \arrow[rr, bend right, "h_{Q,O} \circ h_{R,Q}"'] & M \arrow[r, tail, "h_{M,O}"] & O
\end{tikzcd}$$

Once more, we rephrase the method using wrappers.

Proposition 4.15 (Concurrent Minimization)**.** *If for $\alpha \colon S \to R$ and $\beta \colon O \to P$, $W_R = (S, Q, \alpha, h_{R,Q})$ is an initialized and closed R-wrapper and $W_O = (Q, P, h_{Q,O}, \beta)$ is a responsive and consistent O-wrapper, then $W_M = (S, P, h_{R,M} \circ \alpha, \beta \circ h_{M,O})$ is a complete M-wrapper.*

Proof. The wrapper W_M reaches M for the same reason as in Proposition 4.13, and it observes M for the same reason as in Proposition 4.14. □

Moreover, the approximated response of W_M can be obtained from those of W_R and W_O simply by composition—superscripting the target automaton of the relevant wrapper, we have $\xi^M = \xi^O \circ \xi^R$. Further, $\xi^M_I = \xi^R_I$, $\xi^M_Y = \xi^O_Y$, and $\xi^M_\delta = \xi^O_\delta \circ \xi^R$.

Concretely, this means we can essentially compose the partial results into the final one. In Table 4.4 we have substituted each row of Figure 4.1a, where the rows are states, by the row from Table 4.3a that is labeled by that state. The resulting hypothesis is again isomorphic to M.

4.1 Conformance Testing

We have shown in this section how the concept of an approximation can be generalized to study uniformly the data structures used in both learning and minimization. We now outline briefly how by a different generalization we can study also the kind of conformance testing where a known automaton is tested for equivalence with a black box system. The generalization consists in studying the same approximation under different target languages.

Consider two minimal automata M and N. We wish to determine whether the known automaton M is equal to the unknown automaton N. In what follows, we fix a single approximation for which any properties, as well as the observation structure and hypothesis will be parameterized by the automaton—M or N—for which the language is considered at that moment. The next result is pivotal for the correctness of conformance testing methods.

Proposition 4.16. *If the approximation is complete for M and either reaches or observes N, then $\mathcal{L}_M = \mathcal{L}_N$ if and only if the observation structure for M coincides with the one for N.*

Proof. If $\mathcal{L}_M = \mathcal{L}_N$, then also $t_{\mathcal{L}_M} = t_{\mathcal{L}_N}$ and $\overline{\mathcal{L}_M} = \overline{\mathcal{L}_N}$. The observation structures for M and N therefore coincide by definition.

Conversely, assume that the observation structures coincide. Since the approximation is complete for M, the hypothesis for M can be constructed. The observation structures coincide, so the hypothesis for N can also be constructed and in fact these hypotheses are equal. Together with the fact that the approximation either reaches or observes N, we find from either Theorem 4.11 or Theorem 4.12 that the approximation is complete for N. By Theorem 4.10 and Proposition 2.14, the coinciding hypotheses accept the same language as both M and N. □

The following concrete correspondences between conformance test suites and observation structures used in learning were originally found by Berg et al. [19]. Let us first explain the W-method due to Vasilevskii and Chow [67, 30]. Consider the DA setting with finite minimal DA M and N and finite sets $S, E \subseteq A^*$ such that $\varepsilon \in S$ and the approximation $\mathsf{T}(S, E)$ is complete for M. If the observation structures for M and N do not coincide, then we may conclude that M and N are inequivalent. Assume these structures do coincide. The approximation at this point may neither reach nor observe N, but note that the image of ξ for N (the hypothesis for

N) is of size $|M|$ because the hypotheses for M and N coincide and are isomorphic to M as the approximation is complete for M. Then the image of $r_N \circ \sigma_S$ is of size at least $|M|$, since $\xi = \pi_E \circ o_N \circ r_N \circ \sigma_S$. For each of the states of N that are not reached by this approximation (i.e., that are not in the image of $r_N \circ \sigma_S$) there must be a path of transitions starting at a state that *is* reached and ending in that state that was not reached, in such a way that this path has a length of at most $|N| - |M|$ transitions.[3] Thus, define

$$X = SA^{\leq(b-|M|)} = \left\{ u \cdot v \mid u \in S, v \in A^{\leq(b-|M|)} \right\},$$

where b is any number such that $b \geq |N|$, which we assume to be given. The approximation $\mathsf{T}(X, E)$ is still complete for M and moreover now reaches N. Applying Proposition 4.16, we can test whether M and N are equivalent by testing whether their observation structures induced by this approximation coincide. Note that if the observation structures for $\mathsf{T}(S, E)$ did not coincide, then neither will they coincide for $\mathsf{T}(X, E)$. Hence, $\mathsf{T}(S, E)$ need not be considered separately at all. Of course, in practice we would start from $\mathsf{T}(S, E)$ and gradually expand to $\mathsf{T}(X, E)$.

The method does not depend on the approximation being an observation table. This means in particular that we can substitute $\mathsf{T}(X, E)$ for a discrimination tree approximation $\mathsf{DT}(X, \tau)$, where $\tau \in \mathsf{DT}_{\mathcal{P}S}$, that observes M. Testing the resulting test suite is an instance of the method described by Lee and Yannakakis [50], which improves on the W-method.

Given an upperbound b on the number of states of the target automaton in learning, we can now realize equivalence queries us-

[3] Because $\varepsilon \in S$ and N is reachable, there exists a path without cycles that starts in a state of N that the approximation reaches and ends in the state that is not reached by the approximation. We can cut the beginning from this path so that in the resulting path only the first state is reached by the approximation. If this path has more than $|N| - |M|$ transitions, then because it has no cycles it passes through more than $|N| - |M|$ distinct states that are not reached by the approximation, but there are at most $|N| - |M|$ of those because the approximation reaches at least $|M|$ states.

ing the concepts that are also being used in learning. Concretely, whenever the hypothesis has been constructed, we can use one of the minimization algorithms described earlier in this section to find an approximation that is complete for the minimization of the hypothesis. We then extend the access strings as in the above definition of the set X and construct observation structures for $\mathcal{L}$ and $\mathcal{L}_H$ simultaneously until a discrepancy is found, which is a counterexample for the learning algorithm.

Ideally, the approximation used in learning should be complete for its own hypothesis, in which case we can immediately make a (temporary) extension by substituting X for S. Optimizations to this integration of learning and testing can then be considered. Much can be said about sufficient conditions for the approximation to be complete for its hypothesis and adjustments to the learning algorithm to ensure these, but we defer a comprehensive treatment of this theory to future work.

5 Moore Automata

It has often been observed that Angluin's learning algorithm and the discrimination tree variation can directly be generalized to an arbitrary output object. The previous sections have carefully presented all definitions and results in such a way that this generalization is immediately obvious. For instance, this is why we insist on defining the final output system as 2^{A^*} rather than $\mathcal{P}(A^*)$; the leaves of the discrimination trees, on the other hand, are taken from $\mathcal{P}S$ because these are not to be subjected to this generalization.

If we change the DA setting by switching from the output set 2 to an arbitrary output set Y, the resulting automata are known as *Moore automata* [53]. The output set is not required to be finite, but this has no practical consequences: in a finite Moore automaton only a finite number of distinct outputs is actually being used.

If more information about the output set is available, it can be inefficient to use algorithms that have been designed for Moore automata in general. For instance, consider Mealy automata [52], which are like Moore automata except that outputs are associated with transitions rather than states. Jacobs and Silva [46] immediately derived an algorithm by using that Mealy automata *are* Moore automata, but with $Y = B^A$ for some set B. However, their observation table generalization that classifies $(B^A)^{A^*}$ in the set $(B^A)^E$ for a set $E \subseteq A^*$ is very inefficient: each time a column is added to the table one needs $|S||A|^2$ membership queries to update it. This yields a membership query complexity of $\mathcal{O}(k^2n^2 + n \log m)$.[4]

Instead, it would be better to take $E \subseteq A^+ = A^* \setminus \{\varepsilon\}$ and classify into B^E. From the output system isomorphism $(B^A)^{A^*} \cong B^{A^+}$, where B^{A^+} is structured as expected, one sees that B^{A^+} is the final output system. The result is a slight variation on the observation table approximation for Moore automata. The only notable change is the notion of responsiveness—this adapted approximation is respon-

[4] Recall that $k = |A|$ and $n = |M|$; m is the length of the longest counterexample.

sive if and only if for all $s_1, s_2 \in S$ such that $\xi(s_1) = \xi(s_2)$ we also have $\mathcal{L}(s_1 a) = \mathcal{L}(s_2 a)$ for all $a \in A$. Sufficient is that $A \subseteq E$, but in this case initializing $E = A$ leads to a worse membership query complexity: rather than $\mathcal{O}(kn^2 + n \log m)$, the same complexity as obtained for DA, one ends up with $\mathcal{O}(kn^2 + k^2 n + n \log m)$ [64]. Instead, one should check for responsiveness every iteration and add individual violating elements of A to E.

For similar reasons, reusing the Moore discrimination trees for Mealy automata would not be a good idea; instead, the internal nodes should draw experiments from A^+ and distinguish outputs in B:

$$\mathsf{MDT}_L \ ::= \ \mathsf{Leaf}(L) \mid \mathsf{Node}(A^+, \mathsf{DT}_L^B)$$

Again, there are no exciting differences.

In practice, one often encounters systems that release an output after each input. This does not actually change the semantics of our automata; we simply know the output associated with each prefix of any queried word. Advantage should be taken of this additional knowledge, which may have additional distinguishing power and thereby can save queries. This approach was taken for observation tables by Niese [55] and for discrimination trees by Isberner [42].

We give an example in the familiar setting of DA. Consider the language of words over $\{a, b\}$ that contain an a and end with a b. A representation of the observation structure of $\mathsf{T}(S, E)$, where $S = \{\varepsilon, a, ab\}$ and $E = \{\varepsilon, a, ba\}$, is shown in Table 5.1a. The corresponding extended observation table is given in Table 5.1b. Formally, it represents the observation structure for the approximation $(S, (2^*)^E, \sigma_S, \bar{\pi}_E \circ \rho)$, where

$$\begin{gathered} \rho\colon 2^{A^*} \to (2^+)^{A^*} \\ \rho(f)(\varepsilon) = f(\varepsilon) \\ \rho(f)(va) = \rho(f)(v) \cdot f(a) \end{gathered} \qquad \begin{gathered} \bar{\pi}_E\colon (2^+)^{A^*} \to (2^+)^E \\ \bar{\pi}_E(f)(e) = f(e). \end{gathered}$$

The properties that allow for the construction of the hypothesis are as expected.

	ε	a	ba
ε	0	0	0
a	0	0	0
ab	1	0	0
b	0	0	0
aa	0	0	0
aba	0	0	0
abb	1	0	0

(a) Normal table

	ε	a	ba
ε	0	00	000
a	0	00	010
ab	1	10	110
b	0	00	000
aa	0	00	010
aba	0	00	010
abb	1	10	110

(b) Extended table

Table 5.1: Observation tables for the language of words over $\{a, b\}$ that contain an a and end with a b

Clearly, if for some $s_1, s_2 \in S$ we have $\xi(s_1) = \xi(s_2)$ in the extended table, then the same equality holds in the normal table. The converse is not true: in the example we have $\xi(\varepsilon) = \xi(a)$ in the normal table, but not in the extended table. The inequality actually happens to reveal itself in the normal table as an inconsistency: we have $\xi_\delta(\varepsilon)(b) \neq \xi_\delta(a)(b)$. Conversely, the extended table may have inconsistencies not present in the normal table. Furthermore, it may have closedness defects that the normal table does not have: if in our example we would reduce S to $\{\varepsilon\}$, then the normal table would be closed, but this is not the case for the extended table.

6 Algebraic Structure

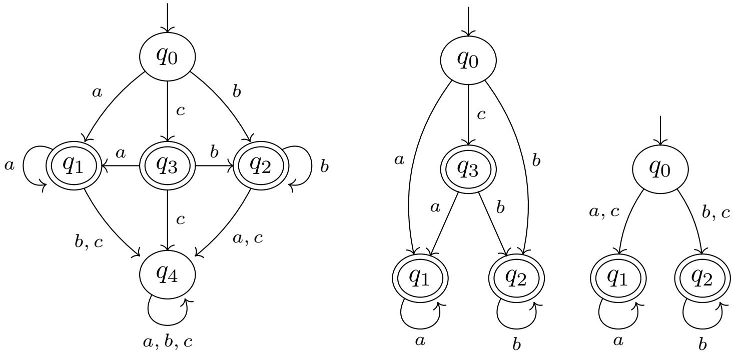
Minimal automata can often be compressed still further after noticing that the state space exhibits certain algebraic structure. For instance, if the language of each state can be described as the union of languages accepted by some significantly small subset of the states, it pays off to use the well-known formalism of *nondeterministic automata* (NDA). Here one runs the automaton by keeping track of a set of states, and such a set is accepting if any of its states accepts. To this end, the automaton model allows for any number of initial states, and each state may proceed after reading an input symbol to a set of successor states.[5]

A more basic structure is observed in *partial automata* (PA), which just have an implicit *sink state* that accepts the empty language. A simple example of both kinds of compression is given in Figure 6.1.

In both of these cases there is an implicit state space, which is obtained by transforming the original set of states. More precisely, it can be described by a functor $T\colon \mathbf{Sets} \to \mathbf{Sets}$. For NDA this is the power set functor $\mathcal{P}$; for PA it is the functor $(-) \uplus 1$ that assigns to each set X the set $\{(x, 0) \mid x \in X\} \cup \{(*, 1)\}$. There are two important operations associated with these functors. First, the automata should be at least as expressive as DA, for which we need a transformation $X \to TX$ for each set X. For instance, in the case of the power set functor we would map each $x \in X$ to the singleton $\{x\}$. Second, for NDA we have defined the implicit state space to be the power set of the set of states, but after reading an input symbol every current state moves to a set of states. The resulting set of sets of states needs to be collapsed into a single set of states. In general, we thus need an additional transformation $TTX \to TX$, which for the power set takes the union of a set of sets.

These transformations need to be well-behaved and interact ap-

[5] A popular interpretation is that a single successor is chosen—hence the “nondeterminism” in the name—but this is detrimental to the semantics that should be kept in mind.



(a) DA **(b)** PA **(c)** NDA

Figure 6.1: Automata accepting the language over the alphabet $\{a, b, c\}$ described by the regular expression $aa^* + bb^* + ca^* + cb^*$. Regarding the minimal DA on the left, we notice that q_3 accepts the union of the languages accepted by q_1 and q_2. Furthermore, q_4 accepts the union of the languages accepted by the empty set of states—it accepts the empty language. The NDA resulting from the corresponding compression is shown on the right; with a PA, as shown in the middle, we can only make q_4 implicit.

propriately: we will require the functor T to be a *monad.* Towards a formal definition of this concept, we introduce a key concept in category theory that describes morphisms between functors.

Definition 6.1 (Natural Transformation)**.** Given two functors $F, G\colon \mathbf{C} \to \mathbf{D}$, a *natural transformation* υ from F to G, denoted $\upsilon\colon F \Rightarrow G$, consists of a *component* $\upsilon_C\colon FC \to GC$ in $\mathbf{D}$ at each object C in $\mathbf{C}$, in such a way that for all morphisms $f\colon C \to D$ in $\mathbf{C}$ the diagram below commutes.

$$\begin{tikzcd}
FC \arrow[r, "\upsilon_C"] \arrow[d, "Ff"'] & GC \arrow[d, "Gf"] \\
FD \arrow[r, "\upsilon_D"] & GD
\end{tikzcd}$$

We may leave out subscripts to improve readability.

For each functor $F\colon \mathbf{C} \to \mathbf{D}$ there is an identity natural transformation $\mathsf{ID}_F\colon F \Rightarrow F$ with components $(\mathsf{ID}_F)_C = \mathsf{id}_{FC}$. If $\zeta\colon F \Rightarrow G$ and $\theta\colon G \Rightarrow H$ are natural transformations, then there is a natural transformation $\theta \circ \zeta\colon F \Rightarrow H$ given by $(\theta \circ \zeta)_C = \theta_C \circ \zeta_C$. Thus, we may employ diagrammatic reasoning with natural transformations without having to quantify over their components. For this purpose it is additionally useful to note that given a natural transformation $\upsilon\colon F \Rightarrow G$ between functors $F, G\colon \mathbf{C} \to \mathbf{D}$ we can define for each functor $J\colon \mathbf{E} \to \mathbf{C}$ a natural transformation $\upsilon J\colon FJ \Rightarrow GJ$ with components $(\upsilon J)_E = \upsilon_{JE}$; similarly, given a functor $K\colon \mathbf{D} \to \mathbf{E}$, we have a natural transformation $K\upsilon\colon KF \Rightarrow KG$ with components $(K\upsilon)_C = K(\upsilon_C)$.

Definition 6.2 (Monad)**.** A *monad* (T, η, μ) in a category $\mathbf{C}$ comprises an endofunctor $T\colon \mathbf{C} \to \mathbf{C}$ and two natural transformations

$$\eta\colon \mathsf{Id}_{\mathbf{C}} \Rightarrow T \qquad\qquad \mu\colon TT \Rightarrow T$$

subject to commutativity of the following diagrams.

$$\begin{tikzcd}
T \arrow[r, Rightarrow, "\eta T"] \arrow[d, Rightarrow, "T\eta"'] \arrow[dr, Rightarrow, "\mathsf{ID}_T"] & TT \arrow[d, Rightarrow, "\mu"] \\
TT \arrow[r, Rightarrow, "\mu"] & T
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
TTT \arrow[r, Rightarrow, "\mu T"] \arrow[d, Rightarrow, "T\mu"'] & TT \arrow[d, Rightarrow, "\mu"] \\
TT \arrow[r, Rightarrow, "\mu"] & T
\end{tikzcd}$$

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

The power set monad will be our motivating example throughout this section. As suggested before, η and μ are for this case defined by $\eta_U(u) = \{u\}$ and $\mu_U(X) = \bigcup X$, where $u \in U$ and $X \in \mathcal{P}\mathcal{P}U$. To see that these are natural, note that for all functions $f\colon U \to V$ and $u \in U$,

$$(\eta_V \circ f)(u) = \{f(u)\} = \mathcal{P}f(\{u\}) = (\mathcal{P}f \circ \eta_U)(u)$$

and for each $X \in \mathcal{P}\mathcal{P}U$,

$$\begin{aligned}
(\mathcal{P}f \circ \mu_U)(X) &= \mathcal{P}f\left(\bigcup X\right) \\
&= \{f(u) \mid u \in \bigcup X\} \\
&= \{f(u) \mid u \in x \wedge x \in X\} \\
&= \bigcup\{\{f(u) \mid u \in x\} \mid x \in X\} \\
&= \bigcup\{\mathcal{P}f(x) \mid x \in X\} \\
&= \bigcup \mathcal{P}\mathcal{P}f(X) \\
&= (\mu_V \circ \mathcal{P}\mathcal{P}f)(X).
\end{aligned}$$

The monad laws assert that for each $X \in \mathcal{P}U$ we have

$$\bigcup\{\{x\} \mid x \in X\} = X = \bigcup\{X\}$$

and that taking unions is associative, i.e., for each $X \in \mathcal{P}\mathcal{P}\mathcal{P}U$,

$$\bigcup\left\{\bigcup V \mid V \in X\right\} = \bigcup\bigcup X.$$

These follow from the set theoretical axiom of union.

At this point, we can already define automata with state spaces enriched by a monad.

Definition 6.3 (T-automaton)**.** A T*-automaton* for a monad T is an object Q in $\mathbf{B}$ equipped with morphisms

$$\mathsf{init}^\dagger_Q \colon I \to TQ \qquad \mathsf{out}^\dagger_Q \colon Q \to Y \qquad \delta^\dagger_Q \colon Q \to \mathcal{D}TQ.$$

Although with some effort these T-automata could be described as automata in a category as per the general definition from Section 2,[6] this category does not always have a suitable factorization system. For instance, there is in general no unique minimal NDA for a given language. To still obtain the kind of minimization intended by our motivating examples, we turn to an actually larger category that is also associated with the monad.

Definition 6.4 (Eilenberg–Moore Category)**.** An *algebra for a monad* (T, η, μ), or just a T-algebra[7], in a category $\mathbf{C}$ is a tuple (X, x) of an object X and a morphism $x \colon TX \to X$ in $\mathbf{C}$ making the diagrams below commute.

$$\begin{tikzcd}
X \arrow[r, "\eta"] \arrow[dr, "\mathsf{id}"'] & TX \arrow[d, "x"] \\
& X
\end{tikzcd}
\qquad
\begin{tikzcd}
TTX \arrow[r, "Tx"] \arrow[d, "\mu"'] & TX \arrow[d, "x"] \\
TX \arrow[r, "x"] & X
\end{tikzcd}$$

A homomorphism from (U, u) to (V, v), both algebras for T, is a morphism $f \colon U \to V$ in $\mathbf{C}$ making the diagram below commute.

$$\begin{tikzcd}
TU \arrow[r, "Tf"] \arrow[d, "u"'] & TV \arrow[d, "v"] \\
U \arrow[r, "f"] & V
\end{tikzcd}$$

[6] The category intended here is the *Kleisli* category for the monad T. However, for this to work Definition 6.3 would need two adjustments. First, Y would have to be of the form TZ for some Z in $\mathbf{B}$. Second, we would have to model the dynamics algebraically ($\delta^\dagger \colon \mathcal{D}Q \to TQ$).

[7] Here and hereafter we disregard the notion of a T-algebra defined in Section 2.1 for the endofunctor T.

This yields the *Eilenberg–Moore* category $\mathbf{EM}(T)$ of T-algebras and their homomorphisms.

An algebra for the power set monad is a tuple (K, κ), where $\kappa\colon \mathcal{P}K \to K$ satisfies for each $k \in K$, $\kappa(\{k\}) = k$; and for all $V \in \mathcal{P}\mathcal{P}K$,

$$\kappa\left(\bigcup V\right) = \kappa(\{\kappa(W) \mid W \in V\}).$$

We prefer a less cumbersome notation. Instead of giving a name to this algebra map κ, we will usually say that K *is* a $\mathcal{P}$-algebra and denote for each $U \in \mathcal{P}K$ the application of its algebra map to U by $\bigsqcup U$. Thus, the above requirements become

$$\bigsqcup\{k\} = k \qquad \bigsqcup\left(\bigcup V\right) = \bigsqcup\left\{\bigsqcup W \mid W \in V\right\}.$$

The intended use of $\bigsqcup$ will be clear from the context.

Given $\mathcal{P}$-algebras K and L, $f\colon K \to L$ is a $\mathcal{P}$-algebra homomorphism if and only if it satisfies

$$f\left(\bigsqcup U\right) = \bigsqcup\{f(k) \mid k \in U\}.$$

for every $U \in \mathcal{P}K$.

The category $\mathbf{EM}(\mathcal{P})$ is essentially the category of *complete semilattices* and their homomorphisms. Regarding this, we only note that each $\mathcal{P}$-algebra K is equipped with a partial order $\sqsubseteq$ for which least upper bounds of arbitrary subsets $U \in \mathcal{P}K$ are given by $\bigsqcup U$: for all $k, k' \in K$, $k \sqsubseteq k'$ if and only if $\bigsqcup\{k, k'\} = k'$.

Note that for each object C in $\mathbf{C}$ there is a *free algebra* (TC, μ) in $\mathbf{EM}(T)$: the algebra laws in this case are satisfied by the monad laws. For instance, for any set X, the set $\mathcal{P}X$ has a $\mathcal{P}$-algebra structure given by taking unions.

Suppose we have instantiated the abstract theory of Section 2 in some base category $\mathbf{B}$. We fix a monad (T, η, μ) on $\mathbf{B}$ and aim to lift the assumptions to $\mathbf{EM}(T)$. That is, $\mathbf{EM}(T)$ will now be our base category; we will refer to $\mathbf{B}$ as the *original* base category. In $\mathbf{EM}(T)$, we consider as the initial state object the free algebra (TI, μ), and we assume that Y and $\mathcal{D}$ can be lifted to $\mathbf{EM}(T)$. That is, we assume an algebra (Y, y) for T and a *distributive law* $\rho\colon TD \to DT$.

Definition 6.5 (Distributive Law)**.** A *distributive law* of the monad T over the endofunctor D is a natural transformation $\rho\colon TD \Rightarrow DT$ rendering the diagrams below commutative.

$$\begin{tikzcd}
\mathcal{D} \arrow[r, Rightarrow, "\eta\mathcal{D}"] \arrow[dr, Rightarrow, "\mathcal{D}\eta"'] & T\mathcal{D} \arrow[d, Rightarrow, "\rho"] \\
 & \mathcal{D}T
\end{tikzcd}
\qquad
\begin{tikzcd}
TT\mathcal{D} \arrow[r, Rightarrow, "T\rho"] \arrow[d, Rightarrow, "\mu"'] & T\mathcal{D}T \arrow[r, Rightarrow, "\rho"] & \mathcal{D}TT \arrow[d, Rightarrow, "\mathcal{D}\mu"] \\
T\mathcal{D} \arrow[rr, Rightarrow, "\rho"] & & \mathcal{D}T
\end{tikzcd}$$

For the power set monad, there is some variation possible as to how the algebra map $y\colon \mathcal{P}2 \to 2$ is defined. These variations will be reflected in the formal semantics that will become associated with our T-automata. The algebra laws demand only that $\bigsqcup\{0\} = 0$ and $\bigsqcup\{1\} = 1$. If we define $\bigsqcup\emptyset = 0$, then the empty set of states yields an implicit sink state; if we define $\bigsqcup\emptyset = 1$, then the empty set implicates a state that accepts everything. If we define $\bigsqcup 2 = 0$, then a set of states that contains both accepting and rejecting states is itself considered to be rejecting; if we define $\bigsqcup 2 = 1$, then a set of states that contains both accepting and rejecting states is itself considered to be accepting. Thus, the definition

$$\bigsqcup\emptyset = 0 \qquad\qquad \bigsqcup 2 = 1$$

corresponds to nondeterministic automata; the T-automata that will correspond to

$$\bigsqcup\emptyset = 1 \qquad\qquad \bigsqcup 2 = 0$$

are known as *universal automata.*

A distributive law of the power set monad over $(-)^A$ is given by the components

$$\begin{gathered}
\rho_X\colon \mathcal{P}(X^A) \to (\mathcal{P}X)^A \\
\rho_X(U)(a) = \{f(a) \mid f \in U\}.
\end{gathered}$$

The proof is left as an exercise.

In general, the lifted functor associated with a distributive law ρ is $\widehat{\mathcal{D}}\colon \mathbf{EM}(T) \to \mathbf{EM}(T)$ given by $\widehat{\mathcal{D}}(X, x) = (\mathcal{D}X, \mathcal{D}x \circ \rho)$ on algebras and $\widehat{\mathcal{D}}f = \mathcal{D}f$ on algebra homomorphisms. For distributive law provided for the power set, we obtain for all $U \subseteq X^A$ and $a \in A$,

$$\left(\bigsqcup U\right)(a) = \bigsqcup\{f(a) \mid f \in U\}.$$

We can now present an automaton in $\mathbf{EM}(T)$ as a T-algebra (Q, q) endowed with the following structure:

$$\begin{tikzcd}
(TI, \mu) \arrow[r, "\mathsf{init}"] & (Q, q) \arrow[r, "\mathsf{out}"] \arrow[d, "\delta"] & (Y, y) \\
 & \widehat{\mathcal{D}}(Q, q) &
\end{tikzcd}$$

There are a few basic T-algebra homomorphisms that we assume known to keep the upcoming proofs concise.

Lemma 6.6. *The following are T-algebra homomorphisms:*

- $Tf\colon (TU, \mu) \to (TV, \mu)$ *for any function* $f\colon U \to V$*;*
- $x\colon (TX, \mu) \to (X, x)$ *for any T-algebra* (X, x)*; and*
- $\rho_X\colon (T\mathcal{D}X, \mu) \to \widehat{\mathcal{D}}(TX, \mu)$ *for any set* X*.*

Proof. Each of these corresponds to an elementary equation. For the first, it is naturality of μ; the second depends on the interaction x is required to have with μ; the last simply asserts the second equation satisfied by the distributive law ρ. □

The following is well known.

Proposition 6.7. *Given an object U in $\mathbf{C}$ and an algebra (V, v) for T, there is a bijective correspondence between morphisms*

$$\frac{U \to V \qquad \textit{in } \mathbf{B}}{(TU, \mu) \to (V, v) \quad \textit{in } \mathbf{EM}(T).}$$

Proof. Given $f\colon U \to V$ in $\mathbf{B}$, we define $f^\sharp = v \circ Tf$; given $g\colon (TU, \mu) \to (V, v)$ in $\mathbf{EM}(T)$, we define $g^\dagger = g \circ \eta_U$. This yields $f^{\sharp\dagger} = v \circ Tf \circ \eta_U$ and $g^{\dagger\sharp} = v \circ T(g \circ \eta_U)$. For bijectivity, we thus need to show

$$v \circ Tf \circ \eta_U = f \qquad\qquad v \circ T(g \circ \eta_U) = g,$$

which follow quite directly from naturality of η and monad and algebra laws:

$$\begin{tikzcd}
U \arrow[r, "f"] \arrow[d, "\eta"'] & V \arrow[d, "\eta"'] \arrow[dr, "\mathsf{id}"] & \\
TU \arrow[r, "Tf"] & TV \arrow[r, "v"] & V
\end{tikzcd}
\qquad
\begin{tikzcd}
TU \arrow[r, "T\eta"] \arrow[dr, "\mathsf{id}"'] & TTU \arrow[r, "Tg"] \arrow[d, "\mu"] & TV \arrow[d, "v"] \\
& TU \arrow[r, "g"] & V
\end{tikzcd}$$

□

For example, given a function $f\colon W \to X$, where W and X are sets and X has a $\mathcal{P}$-algebra structure, we have

$$f^\sharp\colon \mathcal{P}W \to X \qquad\qquad f^\sharp(U) = \bigsqcup\{f(w) \mid w \in U\}.$$

For a $\mathcal{P}$-algebra homomorphism $g\colon \mathcal{P}W \to X$ we have

$$g^\dagger\colon W \to X \qquad\qquad g^\dagger(w) = g(\{w\}).$$

We are now ready to consider the remaining ingredients of our framework in this setting. The fact that the initial state object is free allows us to obtain a free initial input system. A similar approach was taken by Arbib and Manes [13, Section 6].

Proposition 6.8. *The initial input system @ in* $\mathbf{B}$ *yields an initial input system* $(T@, \mu)$ *in* $\mathbf{EM}(T)$.

Proof. We define an input system structure on $\tilde{@} = (T@, \mu)$ as follows:

$$\begin{aligned}
\mathsf{init}_{\tilde{@}}\colon (TI, \mu) &\to \tilde{@} &\qquad\qquad \delta_{\tilde{@}}\colon \tilde{@} &\to \widehat{\mathcal{D}}\tilde{@} \\
\mathsf{init}_{\tilde{@}} &= T\mathsf{init}_{@} & \delta_{\tilde{@}} &= \rho \circ T\delta_{@}
\end{aligned}$$

Consider an arbitrary input system (U, u) in $\mathbf{EM}(T)$. We obtain by initiality of @ in $\mathbf{B}$ a reachability map $r_U \colon @ \to U$ by considering U as the input system in $\mathbf{B}$ indicated below, where $\tilde{U} = (U, u)$.

$$\begin{tikzcd}
I \arrow[rr, "\mathsf{init}_@"] \arrow[dr, "\eta"'] & & @ \arrow[r, "\delta_@"] \arrow[dd, dotted, "r_U"] & \mathcal{D}@ \arrow[dd, dotted, "\mathcal{D}r_U"] \\
& TI \arrow[dr, "\mathsf{init}_{\tilde{U}}"'] & & \\
& & U \arrow[r, "\delta_{\tilde{U}}"] & \mathcal{D}U
\end{tikzcd}$$

Our candidate reachability map in $\mathbf{EM}(T)$ is $r_{\tilde{U}} = r_U^\sharp = u \circ Tr_U$, which is shown with the commutative diagram below to be an input system homomorphism.

$$\begin{tikzcd}
TI \arrow[rr, "T\mathsf{init}_@"] \arrow[dr, "T\eta"] \arrow[ddr, bend right, "\mathsf{id}"'] & & T@ \arrow[r, "T\delta_@"] \arrow[d, "Tr_U"] & T\mathcal{D}@ \arrow[r, "\rho"] \arrow[d, "T\mathcal{D}r_U"] & \mathcal{D}T@ \arrow[d, "\mathcal{D}Tr_U"] \\
& TTI \arrow[r, "T\mathsf{init}_{\tilde{U}}"] \arrow[d, "\mu"] & TU \arrow[r, "T\delta_{\tilde{U}}"] \arrow[d, "u"] & T\mathcal{D}U \arrow[r, "\rho"] & \mathcal{D}TU \arrow[d, "\mathcal{D}u"] \\
& TI \arrow[r, "\mathsf{init}_{\tilde{U}}"] & U \arrow[rr, "\delta_{\tilde{U}}"] & & \mathcal{D}U
\end{tikzcd}$$

Suppose $h \colon (T@, \mu) \to (U, u)$ is any input system homomorphism. From the commutative diagram below it follows by initiality of @ in $\mathbf{B}$ that $h \circ \eta = r_U$; hence, $h = (h \circ \eta)^\sharp = r_U^\sharp = r_{\tilde{U}}$ using Proposition 6.7.

$$\begin{tikzcd}
I \arrow[r, "\mathsf{init}_@"] \arrow[d, "\eta"'] & @ \arrow[rr, "\delta_@"] \arrow[d, "\eta"'] & & \mathcal{D}@ \arrow[dl, "\eta"'] \arrow[d, "\mathcal{D}\eta"] \\
TI \arrow[r, "T\mathsf{init}_@"] \arrow[dr, bend right, "\mathsf{init}_{\tilde{U}}"'] & T@ \arrow[r, "T\delta_@"] \arrow[d, "h"'] & T\mathcal{D}@ \arrow[r, "\rho"] & \mathcal{D}T@ \arrow[d, "\mathcal{D}h"] \\
& U \arrow[rr, "\delta_{\tilde{U}}"'] & & \mathcal{D}U
\end{tikzcd}$$

□

As for the final output system, we elaborate on a proof due to Balan and Kurz [17, (2.6)]. Let us first define for each output system

X in $\mathbf{B}$ an output system structure on TX:

$$\mathsf{out}_{TX} = \ TX \xrightarrow{T\mathsf{out}_X} TY \xrightarrow{\ y\ } Y$$

$$\delta_{TX} = \ TX \xrightarrow{T\delta_X} T\mathcal{D}X \xrightarrow{\ \rho\ } \mathcal{D}TX$$

Lemma 6.9. *Given an output system X in $\mathbf{B}$, the morphisms $\eta_X \colon X \to TX$ and $\mu_X \colon TTX \to TX$ are output system homomorphisms.*

Proof. Diagram chase:

$$\begin{tikzcd}
& & X \arrow[dl, "\mathsf{out}"'] \arrow[dd, "\eta"] \arrow[rr, "\delta"] & & \mathcal{D}X \arrow[ddl, "\eta"'] \arrow[dd, "\mathcal{D}\eta"] \\
& Y \arrow[dl, "\mathsf{id}"'] & & & \\
Y & TY \arrow[l, "y"] & TX \arrow[l, "T\mathsf{out}"] \arrow[r, "T\delta"'] & T\mathcal{D}X \arrow[r, "\rho"'] & \mathcal{D}TX
\end{tikzcd}$$

$$\begin{tikzcd}
& & & TTX \arrow[dl, "TT\mathsf{out}"'] \arrow[ddd, "\mu"] \arrow[r, "TT\delta"] & TT\mathcal{D}X \arrow[ddd, "\mu"] \arrow[r, "T\rho"] & T\mathcal{D}TX \arrow[r, "\rho"] & \mathcal{D}TTX \arrow[ddd, "\mathcal{D}\mu"] \\
& & TTY \arrow[dl, "Ty"'] \arrow[dd, "\mu"] & & & & \\
& TY \arrow[dl, "y"'] & & & & & \\
Y & & TY \arrow[ll, "y"'] & TX \arrow[l, "T\mathsf{out}"'] \arrow[r, "T\delta"] & T\mathcal{D}X \arrow[rr, "\rho"] & & \mathcal{D}TX
\end{tikzcd}$$

□

Lemma 6.10. *If $f \colon U \to V$ is an output system homomorphism in $\mathbf{B}$, then so is $Tf \colon TU \to TV$.*

Proof. Diagram chase:

$$\begin{tikzcd}
& & TU \arrow[dl, "T\mathsf{out}"'] \arrow[dd, "Tf"] \arrow[r, "T\delta"] & T\mathcal{D}U \arrow[dd, "T\mathcal{D}f"] \arrow[r, "\rho"] & \mathcal{D}TU \arrow[dd, "\mathcal{D}Tf"] \\
& TY \arrow[dl, "y"'] \arrow[d, "\mathsf{id}"] & & & \\
Y & TY \arrow[l, "y"] & TV \arrow[l, "T\mathsf{out}"] \arrow[r, "T\delta"'] & T\mathcal{D}V \arrow[r, "\rho"'] & \mathcal{D}TV
\end{tikzcd}$$

□

Lemma 6.11. *If $\tilde{X} = (X, x)$ is an output system in $\mathbf{EM}(T)$, then $x \colon TX \to X$ is an output system homomorphism in $\mathbf{B}$.*

Proof. This follows directly from $\delta_{\tilde{X}}$ and $\mathsf{out}_{\tilde{X}}$ being T-algebra homomorphisms. □

Proposition 6.12. *The final output system Ω in $\mathbf{B}$ lifts to a final output system (Ω, ω) in $\mathbf{EM}(T)$.*

Proof. Define ω using finality as the unique output system homomorphism $T\Omega \to \Omega$ in $\mathbf{B}$.

$$\begin{tikzcd}
& & T\Omega \arrow[r, "T\delta_\Omega"] \arrow[dl, "T\mathsf{out}_\Omega"'] \arrow[dd, dotted, "\omega"] & T\mathcal{D}\Omega \arrow[r, "\rho"] & \mathcal{D}T\Omega \arrow[dd, dotted, "\mathcal{D}\omega"] \\
& TY \arrow[dl, "y"'] & & & \\
Y & & \Omega \arrow[ll, "\mathsf{out}_\Omega"'] \arrow[rr, "\delta_\Omega"] & & \mathcal{D}\Omega
\end{tikzcd}$$

This definition would directly turn out_Ω and δ_Ω into T-algebra homomorphisms, but note that we still need to verify the algebra laws for ω. The first law dictates $\omega \circ \eta_\Omega = \mathsf{id}_\Omega \colon \Omega \to \Omega$. The identity on Ω and ω are output system homomorphisms, and by Lemma 6.9 the same holds for η_Ω. The equation is thus satisfied by finality. The second law says $\omega \circ \mu_\Omega = \omega \circ T\omega \colon TT\Omega \to \Omega$. By Lemma 6.9 and Lemma 6.10 we conclude using finality that (Ω, ω) is an output system in $\mathbf{EM}(T)$.

It remains to demonstrate finality in this setting. Any output system $\tilde{U} = (U, u)$ in $\mathbf{EM}(T)$ may be considered as an output system U in $\mathbf{B}$, so there we obtain an observability map $o_U \colon U \to \Omega$. We are done if we can show that this is a T-algebra homomorphism $(U, u) \to (\Omega, \omega)$. From Lemma 6.10 and Lemma 6.11 we know that $o_U \circ u$ and $\omega \circ T o_U$ are both output system homomorphisms $TU \to \Omega$, which by finality completes the proof. □

Together, we obtain the following situation for an automaton

$\tilde{Q} = (Q, q)$ in $\mathbf{EM}(T)$.

$$\begin{tikzcd}
(T@, \mu) \arrow[r, "r_Q^\sharp"] \arrow[rr, bend right, "t_{\mathcal{L}_{\tilde{Q}}}"'] & \tilde{Q} \arrow[r, "o_Q"] & (\Omega, \omega)
\end{tikzcd}$$

Let us now examine the different language representations in this setting more closely.

Proposition 6.13. *For any language $\mathcal{L}\colon (T@, \mu) \to (Y, y)$ in $\mathbf{EM}(T)$, the total response is given by $t_{\mathcal{L}} = t_{\mathcal{L}^\dagger}^\sharp$.*

Proof. Recall that the total response is the observability map of the initial input system equipped with a language as its output map (as shown in the proof of Proposition 2.10). Proposition 6.12 shows that we actually obtain $t_{\mathcal{L}}$ by finality of Ω in $\mathbf{B}$:

$$\begin{tikzcd}
 & T@ \arrow[r, "T\delta_@"] \arrow[dl, "\mathcal{L}"'] \arrow[d, dotted, "t_{\mathcal{L}}"] & T\mathcal{D}@ \arrow[r, "\rho"] & \mathcal{D}T@ \arrow[d, dotted, "\mathcal{D}(t_{\mathcal{L}})"] \\
Y & \Omega \arrow[l, "\mathsf{out}_\Omega"'] \arrow[rr, "\delta_\Omega"] & & \mathcal{D}\Omega
\end{tikzcd}$$

Now note that $t_{\mathcal{L}} = t_{\mathcal{L}^\dagger}^\sharp$ if and only if $t_{\mathcal{L}}^\dagger = t_{\mathcal{L}^\dagger}$ because the operations $(-)^\dagger$ and $(-)^\sharp$ are inverse to each other. We show that by finality $t_{\mathcal{L}}^\dagger = t_{\mathcal{L}^\dagger}$, or rather, filling in the definition of $(-)^\dagger$, $t_{\mathcal{L}} \circ \eta = t_{\mathcal{L}\circ\eta}$:

$$\begin{tikzcd}
 & @ \arrow[rr, "\delta_@"] \arrow[d, "\eta"'] & & \mathcal{D}@ \arrow[d, "\mathcal{D}\eta"] \arrow[dl, "\eta"'] \\
 & T@ \arrow[r, "T\delta_@"] \arrow[dl, "\mathcal{L}"'] \arrow[d, "t_{\mathcal{L}}"] & T\mathcal{D}@ \arrow[r, "\rho"] & \mathcal{D}T@ \arrow[d, "\mathcal{D}(t_{\mathcal{L}})"] \\
Y & \Omega \arrow[l, "\mathsf{out}_\Omega"'] \arrow[rr, "\delta_\Omega"] & & \mathcal{D}\Omega
\end{tikzcd}$$

□

Proposition 6.14. *For any language $\mathcal{L}\colon (T@, \mu) \to (Y, y)$ in $\mathbf{EM}(T)$, the alternative language representation is given by*

$$\overline{\mathcal{L}}\colon (TI, \mu) \to (\Omega, \omega) \qquad\qquad \overline{\mathcal{L}} = \overline{\mathcal{L}^\dagger}^\sharp.$$

Proof. Using the initial state map from Proposition 6.8 in the proof of Proposition 2.10, we read that $\overline{\mathcal{L}} = t_{\mathcal{L}} \circ T\mathsf{init}_{@}$; in $\mathbf{B}$ we have $\overline{\mathcal{L}^\dagger} = t_{\mathcal{L}^\dagger} \circ \mathsf{init}_{@}$. By Proposition 6.13, $t_{\mathcal{L}} = t_{\mathcal{L}^\dagger}^\sharp = \omega \circ Tt_{\mathcal{L}^\dagger}$, so

$$\overline{\mathcal{L}} = t_{\mathcal{L}} \circ T\mathsf{init}_{@} = \omega \circ Tt_{\mathcal{L}^\dagger} \circ T\mathsf{init}_{@} = (t_{\mathcal{L}^\dagger} \circ \mathsf{init}_{@})^\sharp = \overline{\mathcal{L}^\dagger}^\sharp. \qquad \square$$

For the case of the power set monad over the DA setting, let us fix the algebra map on 2 to be the one corresponding to nondeterministic automata: for all $U \in \mathcal{P}2$,

$$\bigsqcup U = \begin{cases} 1 & \text{if } 1 \in U \\ 0 & \text{if } 1 \notin U. \end{cases}$$

This yields the following algebra map on the set of languages: for $U \in \mathcal{P}(2^{A^*})$ and $v \in A^*$,

$$\left(\bigsqcup U\right)(v) = \begin{cases} 1 & \text{if } l(v) = 1 \text{ for some } l \in U \\ 0 & \text{if } l(v) = 0 \text{ for all } l \in U. \end{cases}$$

A given language in this setting is a $\mathcal{P}$-algebra homomorphism $\mathcal{L} \colon (\mathcal{P}A^*, \mu) \to (2, y)$. It corresponds by Proposition 6.7 to a function $\mathcal{L}^\dagger \colon A^* \to 2$, of which we think as the actual target language. Interpreting the correspondence using the above definition, we have for each set of words $U \in \mathcal{P}A^*$,

$$\mathcal{L}(U) = \begin{cases} 1 & \text{if there is a } u \in U \text{ such that } \mathcal{L}^\dagger(u) = 1 \\ 0 & \text{if there is no } u \in U \text{ such that } \mathcal{L}^\dagger(u) = 1. \end{cases}$$

From Proposition 6.13 we furthermore read that for all $U \in \mathcal{P}A^*$ and $v \in A^*$,

$$t_{\mathcal{L}}(U)(v) = \begin{cases} 1 & \text{if there is a } u \in U \text{ such that } \mathcal{L}^\dagger(uv) = 1 \\ 0 & \text{if there is no } u \in U \text{ such that } \mathcal{L}^\dagger(uv) = 1. \end{cases}$$

Finally, $\overline{\mathcal{L}} \colon \mathcal{P}1 \to 2^{A^*}$ is given by

$$\overline{\mathcal{L}}(\emptyset)(v) = 0 \qquad\qquad \overline{\mathcal{L}}(1)(v) = \mathcal{L}^\dagger(v)$$

for $v \in A^*$.

The proofs of Proposition 6.8 and Proposition 6.12 further allow us to interpret reachability and observability in $\mathbf{EM}(\mathcal{P})$. Given an automaton Q in $\mathbf{EM}(\mathcal{P})$ as below on the left (with the algebra structure left implicit), consider also the DA on the right obtained by forgetting the algebra structure and applying Proposition 6.7 to its initial state map.

$$\begin{tikzcd}
\mathcal{P}I \arrow[r, "\mathsf{init}"] & Q \arrow[r, "\mathsf{out}"] \arrow[d, "\delta"] & 2 \\
 & Q^A &
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
I \arrow[r, "\mathsf{init}^\dagger"] & Q \arrow[r, "\mathsf{out}"] \arrow[d, "\delta"] & 2 \\
 & Q^A &
\end{tikzcd}$$

Because observability maps in $\mathbf{EM}(\mathcal{P})$ are just observability maps in **Sets**, these two automata have the same notion of observability; reachability, on the other hand, is weaker for the automaton on the left: the automaton is reachable if for each state $q \in Q$ there is a set of words $U \in \mathcal{P}(A^*)$ such that $\bigsqcup\{r(u) \mid u \in U\} = q$, where $r\colon A^* \to Q$ is the reachability map in **Sets** of the automaton on the right (see the proof of Proposition 6.8).

Back in the abstract setting, the definition of $\widehat{\mathcal{D}}$ lifts our assumption about $\mathcal{D}$ to $\mathbf{EM}(T)$ in the following sense: the functor $\widehat{\mathcal{D}}$ maps each T-algebra homomorphism that is monic in $\mathbf{B}$ to a T-algebra homomorphism that is monic in $\mathbf{B}$. We now further assume that T preserves $\mathcal{E}$, which will give us a factorization system in $\mathbf{EM}(T)$. Moreover, Lemma 2.17 can be lifted directly.

Lemma 6.15. *Consider commutative diagrams in* $\mathbf{B}$ *as below.*

$$\begin{tikzcd}
C \arrow[d, "f"'] \arrow[dr, "w"] & \\
D \arrow[r, tail, "u"] & E
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
K \arrow[r, two heads, "v"] \arrow[dr, "x"'] & L \arrow[d, "g"] \\
 & Z
\end{tikzcd}$$

If (C,c), (D,d), and (E,e) are T-algebras such that w and u are T-algebra homomorphisms, then f is a T-algebra homomorphism;

if (K,k)*,* (L,l)*, and* (Z,z) *are* T*-algebras such that* x *and* v *are* T*-algebra homomorphisms, then* g *is a* T*-algebra homomorphism.*

Proof. We prove $f \circ c = d \circ Tf$ and $g \circ l = k \circ Tg$ by composing with the mono u and precomposing with the epi Tv, respectively:

$$\begin{tikzcd}
TC \arrow[r, "c"] \arrow[d, "Tf"'] \arrow[dr, "Tw"] & C \arrow[r, "f"] \arrow[dr, "w"'] & D \arrow[d, tail, "u"] \\
TD \arrow[r, "Tu"'] \arrow[drr, bend right, "w"'] & TE \arrow[r, "e"'] & E \\
& & D \arrow[u, tail, "u"']
\end{tikzcd}
\qquad
\begin{tikzcd}
TL \arrow[rr, bend left, "l"] & & \\
TK \arrow[u, two heads, "Tv"] \arrow[r, "k"] \arrow[d, two heads, "Tv"'] \arrow[dr, "Tx"] & K \arrow[r, "v"] \arrow[dr, "x"'] & L \arrow[d, "g"] \\
TL \arrow[r, "Tg"'] & TZ \arrow[r, "z"'] & Z
\end{tikzcd}$$

□

Proposition 6.16. *The Eilenberg–Moore category* $\mathbf{EM}(T)$ *has a factorization system inherited from* $\mathbf{B}$*. Specifically, it is given by*

$$(\{e \in \mathcal{E} \mid e \textit{ is a } T\textit{-algebra homomorphism}\},$$
$$\{m \in \mathcal{M} \mid m \textit{ is a } T\textit{-algebra homomorphism}\}).$$

Proof. As in the proof of Proposition 2.20, the properties (F1) through (F3) are inherited directly. For (F4), consider a T-algebra homomorphism $h\colon (U,u) \to (V,v)$. We factorize $h\colon U \to V$ in $\mathbf{B}$ as $h = j \circ i$ through X and define $x\colon TX \to X$ as the diagonal indicated below on the left, using that T preserves $\mathcal{E}$.

$$\begin{tikzcd}
TU \arrow[r, two heads, "Ti"] \arrow[d, "u"'] & TX \arrow[d, "Tj"] \arrow[ddl, dotted, "x"'] \\
U \arrow[d, "i"'] & TV \arrow[d, "v"] \\
X \arrow[r, tail, "j"'] & V
\end{tikzcd}
\qquad
\begin{tikzcd}
TU \arrow[r, "Ti"] \arrow[d, "u"'] \arrow[dr, "Th"] & TX \arrow[d, "Tj"] \\
U \arrow[d, "i"'] \arrow[dr, "h"] & TV \arrow[d, "v"] \\
X \arrow[r, "j"'] & V
\end{tikzcd}$$

Commutativity of the rectangle on the left is indicated on the right. By the definition of x, i and j will be T-algebra homomorphisms. However, it remains to verify the algebra laws for x. We prove both

$x \circ \eta_X = \mathsf{id}_X$ and $x \circ \mu = x \circ Tx$ by composing with the mono j:

$$\begin{tikzcd}
X \arrow[r, tail, "j"] \arrow[d, "\eta"'] & V \arrow[d, "\eta"'] \arrow[dd, bend left, "\mathsf{id}"] \\
TX \arrow[r, "Tj"] \arrow[d, "x"'] & TV \arrow[d, "v"'] \\
X \arrow[r, tail, "j"] & V
\end{tikzcd}
\qquad
\begin{tikzcd}
TTX \arrow[drr, "TTj"'] \arrow[dd, "Tx"'] \arrow[drrr, bend left, "\mu"] & & & \\
& & TTV \arrow[d, "Tv"'] \arrow[dr, "\mu"] & TX \arrow[d, "Tj"'] \arrow[ddr, bend left, "x"] & \\
TX \arrow[rr, "Tj"] \arrow[d, "x"'] & & TV \arrow[dr, "v"'] & TV \arrow[d, "v"] & \\
X \arrow[rrr, tail, "j"] & & & V & X \arrow[l, tail, "j"']
\end{tikzcd}$$

Finally, consider for (F5) a commuting square of T-algebra homomorphisms as below on the left,

$$\begin{tikzcd}
(U,u) \arrow[r, two heads, "i"] \arrow[d, "f"'] & (V,v) \arrow[d, "g"] \\
(W,w) \arrow[r, tail, "j"] & (X,x)
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
U \arrow[r, two heads, "i"] \arrow[d, "f"'] & V \arrow[d, "g"] \arrow[dl, dotted, "d"'] \\
W \arrow[r, tail, "j"] & X
\end{tikzcd}$$

where $i \in \mathcal{E}$ and $j \in \mathcal{M}$. In $\mathbf{B}$ this gives us a diagonal d as shown on the right. By Lemma 6.15, d is a T-algebra homomorphism. □

Recall that T-automata implicitly have a state space generated by applying the monad T. We will see that, conversely, given an automaton with a freely generated statespace, we can recover a T-automaton.

Definition 6.17 (Implicit automaton)**.** An *implicit automaton* in $\mathbf{EM}(T)$ is an automaton with a state object of the form (TQ, μ_Q) for some object $Q \in \mathbf{B}$.

Thanks to Proposition 6.7, an implicit automaton can be represented by an object and three morphisms in $\mathbf{B}$. More precisely, there is a bijective correspondence between implicit automata and

T-automata:

$$\begin{tikzcd}
(TI,\mu) \arrow[r, "\mathsf{init}"] & (TQ,\mu) \arrow[r, "\mathsf{out}"] \arrow[d, "\delta"] & (Y,y) \\
 & \widehat{\mathcal{D}}(TQ,\mu) &
\end{tikzcd}$$

in $\mathbf{EM}(T)$

$$\begin{tikzcd}
I \arrow[d, "\mathsf{init}^{\dagger}"] & Q \arrow[r, "\mathsf{out}^{\dagger}"] \arrow[d, "\delta^{\dagger}"] & Y \\
TQ & \mathcal{D}TQ &
\end{tikzcd}$$

in $\mathbf{B}$

The implicit automaton on the left is implicit given the T-automaton on the right. Explicitly, we have

$$\begin{aligned} \mathsf{init} &= \mathsf{init}^{\dagger\sharp} = \mu \circ T(\mathsf{init}^{\dagger}) \\ \mathsf{out} &= \mathsf{out}^{\dagger\sharp} = y \circ T(\mathsf{out}^{\dagger}) \end{aligned} \qquad \delta = \delta^{\dagger\sharp} = \mathcal{D}\mu \circ \rho \circ T(\delta^{\dagger}).$$

The semantics of the example T-automata discussed informally at the beginning of this section were obtained by keeping this conversion to the implicit automaton in mind.

As mentioned before, a minimal automaton in $\mathbf{EM}(T)$ may not be an implicit automaton. Given an automaton in $\mathbf{EM}(T)$, we would like to find an equivalent implicit automaton that is as small as reasonably possible. This can be achieved by finding an appropriate *scoop* of the original automaton.

Definition 6.18 (Scoop [33, 13])**.** A *scoop* for a T-algebra (U,u) is a tuple (X,i,d) that consists of an object X in $\mathbf{B}$ and morphisms $i\colon X \to U$ and $d\colon U \to TX$ such that $i^{\sharp} \circ d = \mathsf{id}_U$.

$$\begin{tikzcd}
U \arrow[r, "d"] \arrow[drr, bend right, "\mathsf{id}"'] & TX \arrow[r, "Ti"] & TU \arrow[d, "u"] \\
 & & U
\end{tikzcd}$$

For instance, $(U, \mathsf{id}_U, \eta_U)$ is always a scoop for (U,u), as u is required to satisfy $u \circ \eta = \mathsf{id}_U$. Any scoop for an automaton in $\mathbf{EM}(T)$ gives rise to a language-equivalent implicit automaton [13, Proposition 9, Section 7].

Proposition 6.19. *If $\tilde{Q} = (Q, q)$ is an automaton in $\mathbf{EM}(T)$ and (X, i, d) is a scoop for $\tilde{Q}$, then the automaton $\tilde{X} = (TX, \mu)$ implicated below accepts the same language as $\tilde{Q}$.*

$$\mathsf{init}^{\dagger}_{\tilde{X}} = d \circ \mathsf{init}^{\dagger}_{\tilde{Q}} \qquad \mathsf{out}^{\dagger}_{\tilde{X}} = \mathsf{out}_{\tilde{Q}} \circ i \qquad \delta^{\dagger}_{\tilde{X}} = \mathcal{D}d \circ \delta_{\tilde{Q}} \circ i$$

Proof. Using Proposition 2.14, the fact that $\mathcal{L}_{\tilde{X}} = \mathcal{L}_{\tilde{Q}}$ will follow after observing that $i^{\sharp} \colon \tilde{X} \to \tilde{Q}$ is an automaton homomorphism. To this end, we need to show

$$\mathsf{init}_{\tilde{Q}} = i^{\sharp} \circ \mathsf{init}^{\dagger\sharp}_{\tilde{X}} \qquad \mathsf{out}_{\tilde{Q}} \circ i^{\sharp} = \mathsf{out}^{\dagger\sharp}_{\tilde{X}} \qquad \delta_{\tilde{Q}} \circ i^{\sharp} = \mathcal{D}(i^{\sharp}) \circ \delta^{\dagger\sharp}_{\tilde{X}}.$$

Applying $(-)^{\dagger}$ to all equations, these become

$$\mathsf{init}^{\dagger}_{\tilde{Q}} = i^{\sharp} \circ \mathsf{init}^{\dagger}_{\tilde{X}} \qquad \mathsf{out}_{\tilde{Q}} \circ i = \mathsf{out}^{\dagger}_{\tilde{X}} \qquad \delta_{\tilde{Q}} \circ i = \mathcal{D}(i^{\sharp}) \circ \delta^{\dagger}_{\tilde{X}}.$$

The equation for the output maps is directly satisfied by the definition provided for $\mathsf{out}^{\dagger}_{\tilde{X}}$; the others follow using the scoop property $i^{\sharp} \circ d = \mathsf{id}_U$. □

This T-automaton induced by a scoop will be called a *scoop automaton*. We will now show that for $\mathbf{B} = \mathbf{Sets}$ there are always scoop automata of the minimal automaton in $\mathbf{EM}(T)$ having at most as many states as the minimal automaton in $\mathbf{Sets}$. In what follows, let $\tilde{N} = (N, n)$ be the minimal realization of $\mathcal{L} \colon (T@, \mu) \to (Y, y)$ in $\mathbf{EM}(T)$, and let M be the minimal realization of $\mathcal{L}^{\dagger} \colon @ \to Y$ in $\mathbf{B}$.

Proposition 6.20. *If every epi in $\mathbf{B}$ splits, then there exist morphisms $i \colon M \to N$ and $d \colon N \to TM$ in $\mathbf{B}$ such that (M, i, d) is a scoop of (N, n).*

Proof. From Proposition 6.13 we know that $t_{\mathcal{L}} = t^{\sharp}_{\mathcal{L}^{\dagger}}$, or, equivalently, $t_{\mathcal{L}^{\dagger}} = t^{\dagger}_{\mathcal{L}} = t_{\mathcal{L}} \circ \eta$. Hence, there is a diagonal i below on the left.

$$\begin{tikzcd}
@ \arrow[r, two heads, "r_M"'] \arrow[dd, "\eta"'] \arrow[rrd, bend left, "t_{\mathcal{L}^\dagger}"] & M \arrow[dd, dotted, "i"'] \arrow[rd, "o_M"] & \\
& & \Omega \\
T@ \arrow[r, "r_{\tilde{N}}"] \arrow[rru, bend right, "t_{\mathcal{L}}"'] & N \arrow[ru, hook, "o_{\tilde{N}}"] &
\end{tikzcd}
\qquad
\begin{tikzcd}
T@ \arrow[d, "Tr_M"'] \arrow[rd, "T\eta"] \arrow[rrd, bend left, "\mathsf{id}"] & & & \\
TM \arrow[d, "Ti"'] & TT@ \arrow[r, "\mu"'] \arrow[ld, "Tr_{\tilde{N}}"] & T@ \arrow[rd, "r_{\tilde{N}}"] & \\
TN \arrow[rrr, "n"] & & & N
\end{tikzcd}$$

From commutativity of the diagram on the right we read that the morphism $n \circ Ti\colon TM \to N$ is an epi. We define $d\colon N \to TM$ to be its right inverse; that is, $n \circ Ti \circ d = \mathsf{id}_N$. This is exactly what is required for (M, i, d) to be a scoop of (N, n). □

6.1 Automaton Wrappers

Given an automaton $\tilde{Q} = (Q, q)$ in $\mathbf{EM}(T)$, we consider $\tilde{Q}$-wrappers of the form $W = ((TS, \mu), (P, p), \alpha^\sharp, \beta)$, where $\alpha\colon S \to Q$ is a morphism in $\mathbf{B}$ and $\beta\colon \tilde{Q} \to (P, p)$ is a morphism in $\mathbf{EM}(T)$.

Proposition 6.21. *If the hypothesis for W can be constructed and $e\colon TS \to H$ has a right inverse $i\colon H \to TS$ in $\mathbf{B}$, then $(S, e \circ \eta, i)$ is a scoop of the hypothesis.*

Proof. Let (H, h) be the hypothesis. The result is seen directly from commutativity of

$$\begin{tikzcd}
H \arrow[r, "i"] \arrow[rrrdd, bend right, "\mathsf{id}"'] & TS \arrow[r, "T\eta"] \arrow[rd, "\mathsf{id}"'] & TTS \arrow[r, "Te"] \arrow[d, "\mu"] & TH \arrow[dd, "h"] \\
& & TS \arrow[rd, "e"] & \\
& & & H
\end{tikzcd}$$

□

If the original base category is **Sets**, we may remove any $s \in S$ from S if there exists a $U \in T(S \setminus \{s\})$ with $\xi(U) = \xi^\dagger(s)$, for doing so leaves the image of ξ unharmed. The absence of elements

s that can thus be removed was introduced by Angluin et al. [9] as a minimality property for observation tables in certain more specific settings. Importantly, if this minimality holds, then the function $\xi^\dagger$ is injective. We will see that $\xi^\dagger$ is the approximated response of a wrapper in **Sets** for the language $\mathcal{L}^\dagger$. Thus, our scoop will have less "states" than the hypothesis of this wrapper in **Sets** has states, and therefore certainly less than the number of states of the minimal automaton in **Sets**.

Specifically, let W' be the wrapper (S, P, α, β) in $\mathbf{B}$ for the automaton Q obtained by forgetting the algebra structure and applying $(-)^\dagger$ to the initial state map. It turns out that we can conveniently reuse entire observation structure representations that have been designed for $\mathbf{B}$.

Lemma 6.22. *For all sets U, T-algebras (J, j) and (K, k), and morphisms $f\colon U \to J$ in $\mathbf{B}$ and $g\colon (J, j) \to (K, k)$ in $\mathbf{EM}(T)$, we have $g \circ f^\sharp = (g \circ f)^\sharp$.*

Proof. We have $g \circ f^\sharp = g \circ j \circ Tf$ and $(g \circ f)^\sharp = k \circ T(g \circ f)$. Their equality follows directly from g being a T-algebra homomorphism:

$$\begin{tikzcd}
TU \arrow[r, "Tf"] & TJ \arrow[r, "Tg"] \arrow[d, "j"'] & TK \arrow[d, "k"] \\
 & J \arrow[r, "g"] & K
\end{tikzcd}$$

□

Proposition 6.23. *Superscripting the relevant wrapper, the observation structure of W is given by*

$$\begin{aligned}
\xi^W &: (TS, \mu) \to (P, p) & \xi^W &= (\xi^{W'})^\sharp \\
\xi^W_I &: (TI, \mu) \to (P, p) & \xi^W_I &= (\xi^{W'}_I)^\sharp \\
\xi^W_Y &: (TS, \mu) \to (Y, y) & \xi^W_Y &= (\xi^{W'}_Y)^\sharp \\
\xi^W_\delta &: (TS, \mu) \to \widehat{\mathcal{D}}(P, p) & \xi^W_\delta &= (\xi^{W'}_\delta)^\sharp
\end{aligned}$$

Proof. The only interesting case is for ξ^W_I; the others follow directly from Lemma 6.22. Using that lemma, we have

$$\xi^W = \beta \circ \mathsf{init}_{\tilde{Q}} = \beta \circ \mathsf{init}^{\dagger\sharp}_{\tilde{Q}} = (\beta \circ \mathsf{init}^\dagger_{\tilde{Q}})^\sharp = (\xi^{W'})^\sharp.$$

□

Notice, however, that the wrappers W and W' generally come with different notions of hypothesis constructability.

From the following rather trivial result regarding the correspondence of Proposition 6.7, it follows that the definitions of initialization and closedness for our wrapper W can be simplified.

Proposition 6.24. *For all objects U in $\mathbf{B}$ and objects (J, j) and (K, k) and morphisms $f\colon (TU, \mu) \to (J, j)$, $g\colon (TU, \mu) \to (K, k)$, and $h\colon (J, j) \to (K, k)$ in $\mathbf{EM}(T)$, the diagram below on the left commutes if and only if the one on the right commutes.*

$$\begin{tikzcd}
TU \arrow[d, "f"'] \arrow[dr, "g"] & \\
J \arrow[r, "h"'] & K
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
U \arrow[d, "f^\dagger"'] \arrow[dr, "g^\dagger"] & \\
J \arrow[r, "h"'] & K
\end{tikzcd}$$

Proof. From left to right we simply note that $f^\dagger = f \circ \eta$ and $g^\dagger = g \circ \eta$. Conversely, assume the diagram on the right commutes. We have $f = j \circ T(f^\dagger)$ and $g = k \circ T(g^\dagger)$. Below we conclude by using that h is a T-algebra homomorphism.

$$\begin{tikzcd}
TU \arrow[r, "T(f^\dagger)"] \arrow[dr, "T(g^\dagger)"'] & TJ \arrow[r, "j"] \arrow[d, "Th"] & J \arrow[d, "h"] \\
& TK \arrow[r, "k"'] & K
\end{tikzcd}$$

□

Corollary 6.25. *The wrapper W is initialized (closed) if and only if there is a morphism $\mathsf{init}^\dagger$ ($\mathsf{close}^\dagger$) in $\mathbf{B}$ making the diagram below on the left (right) commute.*

$$\begin{tikzcd}
I \arrow[d, dashed, "\mathsf{init}^\dagger"'] \arrow[dr, "\xi_I^\dagger"] & \\
H \arrow[r, tail, "m"'] & P
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
S \arrow[d, dashed, "\mathsf{close}^\dagger"'] \arrow[dr, "\xi_\delta^\dagger"] & \\
\mathcal{D}H \arrow[r, tail, "\mathcal{D}m"'] & \mathcal{D}P
\end{tikzcd}$$

As an example, we show that observation tables from the DA setting can be lifted for the case of the power set monad. This is essentially what was originally done by Bollig et al. [22].

Given finite sets of words $S, E \in A^*$, we define an *observation table approximation* for this setting as

$$\widehat{\mathsf{T}}(S,E) = ((\mathcal{P}S, \mu), (P, p_E), \mathcal{P}\sigma_S, \pi_E),$$

where σ_S and π_E are as before, and the function $p_E \colon \mathcal{P}2^E \to 2^E$ is defined for all $U \in \mathcal{P}2^E$ and $e \in E$ by

$$p_E(U)(e) = \begin{cases} 1 & \text{if } f(e) = 1 \text{ for some } f \in U \\ 0 & \text{if } f(e) = 0 \text{ for all } f \in U. \end{cases}$$

For $e \in E$ and each $f \colon E \to 2$, we have

$$p_E(\{f\})(e) = \begin{cases} 1 & \text{if } f(e) = 1 \\ 0 & \text{if } f(e) = 0 \end{cases} = f(e),$$

and for each $V \in \mathcal{P}\mathcal{P}2^E$,

$$\begin{aligned} p_E\left(\bigcup V\right)(e) &= \begin{cases} 1 & \text{if } f(e) = 1 \text{ for some } f \in U \text{ and } U \in V \\ 0 & \text{if } f(e) = 0 \text{ for all } f \in U \text{ and } U \in V \end{cases} \\ &= \begin{cases} 1 & \text{if } p_E(U)(e) = 1 \text{ for some } U \in V \\ 0 & \text{if } p_E(U)(e) = 0 \text{ for all } U \in V \end{cases} \\ &= p_E(\{p_E(U) \mid U \in V\})(e). \end{aligned}$$

Thus, $(2^E, p_E)$ is a $\mathcal{P}$-algebra, and we can readily see that with this definition of p_E, π_E is a $\mathcal{P}$-algebra homomorphism: the function p_E is a restricted version of ω, with this very restriction being provided by π_E.

Recall from the proof of Proposition 6.8 that the reachability map $r_{(N,n)}$ of the minimal automaton (N, n) in $\mathbf{EM}(T)$ is $r_N^\sharp$, where r_N is the reachability map of N in $\mathbf{B}$ (having initial state map $\mathsf{init}^\dagger_{(N,n)}$). Since

$$r_N^\sharp \circ T\sigma_S = m \circ Tr_N \circ T\sigma_S = m \circ T(r_N \circ \sigma_S) = (r_N \circ \sigma_S)^\sharp,$$

the wrapper $((\mathcal{P}S,\mu),(P,p_E),r_N^\sharp\circ\mathcal{P}\sigma_S,\pi_E\circ o_N)$ associated with the approximation $\widehat{\mathsf{T}}(S,E)$ is $((\mathcal{P}S,\mu),(P,p_E),(r_N\circ\sigma_S)^\sharp,\pi_E\circ o_N)$ and therefore an instance of the wrapper W discussed above. Note that the associated wrapper W' is precisely the wrapper corresponding to the approximation $\mathsf{T}(S,E)$.

From Corollary 6.25 it follows that W is initialized if and only if there is a set $U\in\mathcal{P}S$ such that $\xi(U)=\xi_I^\dagger(*)$; it is closed if and only if for all $s\in S$ and $a\in A$ there is a set $U\in\mathcal{P}S$ such that $\xi(U)=\xi_\delta^\dagger(s)(a)$. Bollig et al. [22] use a definition of closedness for which the obvious verification algorithm is much more efficient than the one following from the property given here. We validate their definition for our framework using a generalized proposition.

Proposition 6.26. *For a set K, a $\mathcal{P}$-algebra L, a $\mathcal{P}$-algebra homomorphism $f\colon\mathcal{P}K\to L$, and an element $l\in L$, there is a set $U\in\mathcal{P}K$ such that $f(U)=l$, if and only if*

$$f(\{k\in K\mid f^\dagger(k)\sqsubseteq l\})=l.$$

Proof. Define $X=\{k\in K\mid f^\dagger(k)\sqsubseteq l\}$ and assume that there is a $U\in\mathcal{P}K$ such that $f(U)=l$. For each $k\in U$, we have

$$\begin{aligned}\bigsqcup\{f^\dagger(k),l\}=\bigsqcup\{f(\{k\}),l\}&=\bigsqcup\{f(\{k\}),f(U)\}\\&=f(\{k\}\cup U)=f(U)=l,\end{aligned}$$

so $f^\dagger(k)\sqsubseteq l$, and it follows that $X\cup U=X$. Then

$$\begin{aligned}f(X)&=f(X\cup U)\\&=f\left(\bigcup(\{U\}\cup\{\{k\}\cup U\mid k\in X\})\right)\\&=\bigsqcup(\{f(U)\}\cup\{f(\{k\}\cup U)\mid k\in X\})\\&\overset{\star}{=}\bigsqcup(\{f(U)\}\cup\{f(U)\mid k\in X\})\\&=\bigsqcup(\{f(U)\})=f(U)=l.\end{aligned}$$

The step marked by $\star$ uses that any $k \in X$ satisfies $f(\{k\}) \sqsubseteq l$ and thus

$$f(\{k\} \cup U) = \bigsqcup\{f(\{k\}), f(U)\} = \bigsqcup\{f(\{k\}), l\} = l = f(U).$$

The other part of the proof is trivial. □

Responsiveness and consistency do not seem to admit such nice optimizations. The approximation $\widehat{\mathsf{T}}(S, E)$ is responsive if and only if for all $U, V \in \mathcal{P}S$ such that $\xi(U) = \xi(V)$ we have $\mathcal{L}(U) = \mathcal{L}(V)$; it is consistent if and only if for such U and V we have $\xi_\delta(U) = \xi_\delta(V)$.

As an example, consider $\widehat{\mathsf{T}}(S, E)$ corresponding to Table 6.2. Since $\xi_\delta^\dagger(aaa)(a) = \xi(\{\varepsilon, a\})$, the approximation is closed. Despite the fact that the approximation is minimal in the sense of Angluin et al. [9], we do have consistency defects: $\xi(\{\varepsilon, a\}) = \xi(\{aa, aaa\})$, but $\xi_\delta(\{\varepsilon, a\})(a) \neq \xi_\delta(\{aa, aaa\})(a)$; $\xi(\{aa\}) = \xi(\{a, aa\})$, but $\xi_\delta(\{aa\})(a) \neq \xi_\delta(\{a, aa\})(a)$. Note that the situation is quite different in $\mathbf{B}$: for this example, $\mathsf{T}(S, E)$ is consistent, but not closed.

Consider the language of words over $\{a\}$ of which the length is not equal to one. The approximation $\widehat{\mathsf{T}}(S, E)$ associated with Table 6.3 is initialized, responsive, and dynamical. In Figure 6.4a is shown the actual hypothesis (H, h) in $\mathbf{EM}(\mathcal{P})$, but without its algebra structure. We label each state $x \in H$ by a set $U \in \mathcal{P}S$ such that $e(U) = x$. The single initial state represents $\mathsf{init}_{(H,h)}^\dagger \colon 1 \to H$. Note that in $\mathbf{EM}(\mathcal{P})$ this automaton is reachable: the sink state is always reached by the empty set of words. The choice of state labels in fact defines a splitting $i \colon H \to \mathcal{P}S$ of $e \colon \mathcal{P}S \to H$ in $\mathbf{Sets}$. Applying Proposition 6.21 yields the scoop automaton depicted in Figure 6.4b.

6.2 Counterexample Analysis

In order to complete the algorithm for this setting, it remains to handle counterexamples. The method explained in Section 3.3 started

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

Table 6.2: Inconsistency in $\mathbf{EM}(\mathcal{P})$

	ε	a
ε	1	0
a	0	1
aa	1	1

Table 6.3: A table that is initialized, responsive, and dynamical in $\mathbf{EM}(\mathcal{P})$

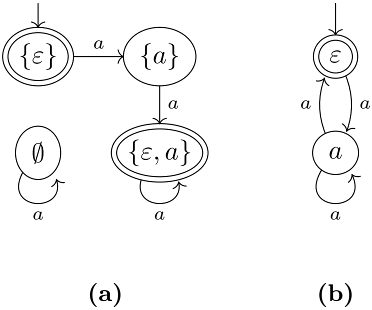


Figure 6.4: Hypothesis for Table 6.3 (a) and its scoop automaton (b) based on the choice of state labels

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

Table 6.5: Initialized, responsive, and dynamical table with e unsplittable

by splitting the function e, which in this case is a $\mathcal{P}$-algebra homomorphism $\mathcal{P}S \to H$. Unfortunately, e may not have a right inverse in $\mathbf{EM}(\mathcal{P})$. For instance, consider the approximation $\widehat{\mathsf{T}}(S, E)$ corresponding to Table 6.5, and suppose $i\colon H \to \mathcal{P}$ is a right inverse of e in $\mathbf{EM}(\mathcal{P})$, i.e., a $\mathcal{P}$-algebra homomorphism such that $e \circ i = \mathsf{id}_H$. There is only one $U \in \mathcal{P}S$ such that $e(U) = e(\{\varepsilon\})$, and there is also just one $U \in \mathcal{P}S$ such that $e(U) = e(\{aa\})$. Therefore, we must have $i(e(\{\varepsilon\})) = \{\varepsilon\}$ and $i(e(\{aa\})) = \{aa\}$. Because $e(\{a\} \cup \{aa\}) = e(\{\varepsilon\} \cup \{aa\})$, it follows that

$$\begin{aligned} i(e(\{a\} \cup \{aa\})) &= i(e(\{\varepsilon\} \cup \{aa\})) = i(e(\{\varepsilon\}) \sqcup e(\{aa\})) \\ &= i(e(\{\varepsilon\})) \cup i(e(\{aa\})) = \{\varepsilon, aa\} \end{aligned}$$

However, there are exactly two sets $U \in \mathcal{P}S$ with $e(U) = e(\{a\})$, namely $\{a\}$ and $\{a, aa\}$, which means that $a \in i(e(\{a\}))$, and since

$$i(e(\{a\} \cup \{aa\})) = i(e(\{a\}) \sqcup e(\{aa\})) = i(e(\{a\})) \cup i(e(\{aa\})),$$

we reach the contradiction $a \in i(e(\{a\} \cup \{aa\}))$.

It was remarked in Section 3.3 that our discussion of counterexample analysis is not exactly categorical. We will actually exploit this in the present setting. It turns out that the algebra structure is almost entirely irrelevant for the processing of a counterexample, the only exception being that when the time comes to expand the approximation, the result must of course be a valid approximation in $\mathbf{EM}(T)$. We will see that the rest of the argument can be carried out in **Sets**. That is, we consider the setting where the original base category $\mathbf{B}$ is **Sets**, and furthermore where $I = \{*\}$ and $\mathcal{D} = (-)^A$ for some set A. Anticipating Section 7.2, the set A need not be finite.

We consider an approximation of the form $((TS, \mu), (P, p), T\sigma, \pi)$, where $\sigma\colon S \to A^*$ is a function and $\pi\colon (Y^{A^*}, \omega) \to (P, \pi)$ is a T-algebra homomorphism. Assuming the hypothesis can be constructed, let (H, ζ) be that hypothesis. A counterexample is a word $z \in A^*$ such that $\mathcal{L}^\dagger_H(z) \neq \mathcal{L}^\dagger(z)$. Suppose the hypothesis has been constructed and we have obtained a counterexample z. The remainder of the argument plays in **Sets**, although we take the relevant concepts (input/output systems, approximations and their observation structures, ...) from $\mathbf{EM}(T)$ by forgetting the algebra structure. Determine a right inverse $i\colon H \to \mathcal{P}S$ of e in **Sets**, which has already been done if the hypothesis was replaced by a scoop automaton suggested by Proposition 6.21. We obtain $h\colon T(A^*) \to \Omega$ as the following composition:

$$\begin{tikzcd}
T(A^*) \arrow[r, "r"] \arrow[rrrr, bend right, dotted, "h"'] & H \arrow[r, "i"] & S \arrow[r, "T\sigma"] & T(A^*) \arrow[r, "t_{\mathcal{L}}"] & \Omega
\end{tikzcd}$$

Note that i, and therefore also h, is not necessarily a T-algebra homomorphism.

We are now ready to present the analogues of Proposition 3.29 and Proposition 3.30.

Proposition 6.27. *If z is a counterexample for the hypothesis (H, ζ) and $\mathcal{L}^\dagger(z) \neq (h \circ T\mathsf{init}_@)^\dagger(*)(z)$,[8] then there are $U \in TS$ and $v \in A^*$ such that $\xi(U) = \xi_I^\dagger(*)$, but $(t_{\mathcal{L}} \circ T\sigma)(U)(v) \neq \overline{\mathcal{L}}^\dagger(*)(v)$.*

Proof. Define $s = (i \circ r_H \circ T\mathsf{init}_@)^\dagger(*)$ and $v = z$. Using Lemma 3.26, we have $\xi(U) = (\xi \circ i \circ r_H \circ T\mathsf{init}_@)^\dagger(*) = \xi_I^\dagger(*)$. Furthermore, the inequality comes down to our assumption:

$$\begin{aligned}(t_{\mathcal{L}} \circ T\sigma)(U)(z) &= (t_{\mathcal{L}} \circ T\sigma \circ i \circ r_H \circ T\mathsf{init}_@)^\dagger(*)(z)\\ &= (h \circ T\mathsf{init}_@)^\dagger(*)(z) \neq \mathcal{L}^\dagger(z) = \overline{\mathcal{L}}^\dagger(*)(z). \qquad \square\end{aligned}$$

Proposition 6.28. *If z is a counterexample for the hypothesis (H, ζ) and $\mathcal{L}^\dagger(z) = (h \circ T\mathsf{init}_@)^\dagger(*)(z)$, then there are $U, U' \in TS$, $a \in A$, and $v \in A^*$ such that $\xi(U) = \xi_\delta(U')(a)$, but*

$$(t_{\mathcal{L}} \circ T\sigma)(U)(v) \neq (\delta_\Omega \circ t_{\mathcal{L}} \circ T\sigma)^\dagger(s)(a)(v).$$

[8] The operation $(-)^\dagger$ was originally defined on certain T-algebra homomorphisms. We use it here just for brevity instead of explicit composition with η, which works on any function of which the domain is a free algebra.

Proof. We prove first that

$$h^\dagger(\varepsilon)(z) \neq h^\dagger(z)(\varepsilon). \tag{6.1}$$

Observe that

$$\begin{aligned}
h^\dagger(\varepsilon)(z) &= (h \circ T\mathsf{init}_@)^\dagger(*)(z) && (\eta(\varepsilon) = \mathsf{init}_@^\dagger(*)) \\
&= \mathcal{L}^\dagger(z) && \text{(assumption)} \\
&\neq \mathcal{L}_H^\dagger(z) && \text{(counterexample)} \\
&= (\mathsf{out}_\Omega \circ h)^\dagger(z) && \text{(Lemma 3.27)} \\
&= h^\dagger(z)(\varepsilon) && \text{(definition of } \mathsf{out}_\Omega\text{).}
\end{aligned}$$

Again, we deduce a breakpoint from (6.1): there are $u, v \in A^*$ and $a \in A$ such that

$$h^\dagger(ua)(v) \neq h^\dagger(u)(av),$$

which may be expressed as a failure of h to be a dynamorphism:

$$(h^A \circ \delta_@)^\dagger(u)(a)(v) \neq (\delta_\Omega \circ h)^\dagger(u)(a)(v). \tag{6.2}$$

Define $U = (i \circ r_H)^\dagger(u)$ and $U' = (i \circ r_H \circ \delta_@^\dagger(u))(a)$. These give us the required inequality

$$\begin{aligned}
&(t_\mathcal{L} \circ T\sigma)(U')(v) \\
&= (t_\mathcal{L} \circ T\sigma \circ i \circ r_H \circ \delta_@^\dagger(u))(a)(v) && \text{(definition of } U'\text{)} \\
&= (h \circ \delta_@^\dagger(u))(a)(v) && \text{(definition of } h\text{)} \\
&= (h^A \circ \delta_@)^\dagger(u)(a)(v) && \text{(definition of } (-)^A\text{)} \\
&\neq (\delta_\Omega \circ h)^\dagger(u)(a)(v) && (6.2) \\
&= (\delta_\Omega \circ t_\mathcal{L} \circ T\sigma \circ i \circ r_H)^\dagger(u)(a)(v) && \text{(definition of } h\text{)} \\
&= (\delta_\Omega \circ t_\mathcal{L} \circ T\sigma)^\dagger(U)(a)(v) && \text{(definition of } U\text{).}
\end{aligned}$$

It remains to show that $\xi(U) = \xi_\delta(s)(a)$. Rewriting these as

$$\xi(U) = (\xi \circ i \circ r_H \circ \delta^\dagger_@(u))(a) = ((\xi \circ i \circ r_H)^A \circ \delta_@)^\dagger(u)(a)$$
$$\xi_\delta(U')(a) = (\xi_\delta \circ i \circ r_H)^\dagger(u)(a),$$

we see that we can conclude by applying Lemma 3.28. □

We evaluate the complexity of the resulting algorithm for our example setting in $\mathbf{EM}(\mathcal{P})$ with the lifted observation tables. Let $k = |A|$ and let n be the size of the minimal DA for the language $\mathcal{L}^\dagger$. Fixing an initialization or a closedness defect increases the image of $\xi^\dagger$. Therefore, we can fix at most n such defects. Fixed consistency defects and the results of processing counterexamples have less generous implications. In general we only know that these have increased the image of ξ. Unfortunately, the image of ξ may be exponential in n. Thus, when the algorithm terminates we will have at most n words in S and at most 2^n words in E. By using a binary search, the number of membership queries needed to process counterexamples is logarithmic in the size m of the longest obtained counterexample, but determining $h^\dagger(u)(v)$ for words $u, v \in A^*$ requires in general a query for each $s \in S$. Altogether, the number of equivalence queries will be in $\mathcal{O}(2^n)$ and the number of membership queries in $\mathcal{O}(k2^n + n2^n \log m)$.

6.3 Discussion

The query complexities that follow directly from our algorithm are rather disappointing, especially considering that we are just learning regular languages here—the original algorithm that learns a DA should in general be much faster. Interestingly, Bollig et al. [22] have come up with an algorithm that is similar to the one developed here, but that does have polynomial query complexities. Given that the minimality property of Angluin et al. [9] holds (there are no $s \in S$ and $U \in \mathcal{P}(S \setminus \{s\})$ such that $\xi^\dagger(s) = \xi(U)$), the hypothesis they construct is an instance of a scoop automaton for our hypothesis. However, their notion of consistency is weaker. In fact, they can construct their hypothesis in some cases when ours is not well-defined. It is this weaker notion of consistency, together with a more intricate argument regarding progress after a counterexample, that ensures a polynomial complexity. For this reason it would be highly interesting to relate their construction properly to our framework.

On the negative side, Bollig et al. observed that their algorithm did not terminate when using the original counterexample processing method due to Angluin [7] (which we do not explain in this thesis). We conjecture that this would be fixed by using our notion of consistency.

A different solution to the complexity analysis is given by Angluin et al. [9]. They replace the factors 2^n by the size of the minimal DA for the reverse of the language under consideration, which is justified by the observation that the number of distinct columns in the table (restricted to its upper part) is bounded by that number. The argument is valid also in our algorithm (for when two unions of rows become distinguished there must have been added a column that was not previously in the table). Note, however, that the minimal DA for the reverse of the language may still be exponentially larger (but also smaller) than the minimal DA for the actual language itself.

The worst case complexities known for the algorithms by Bollig et al. [22] and Angluin et al. [9] are both still worse than the complexities known for the algorithm in the original setting. However, in both cases better performance in practice is reported when the target languages are given by randomly generated regular expressions [22] or NDA [9]; Angluin et al. [9] find that the comparable algorithm that learns a DA performs better when the target languages are given by randomly generated DA. It would be interesting to see how our algorithm relates to their algorithms in practice.

The output object 2 for which we have discussed the learning algorithm can be seen as the free $\mathcal{P}$-algebra $\mathcal{P}(1)$. The algorithm can easily be generalized to any free $\mathcal{P}$-algebra $\mathcal{P}(X)$ for a finite set X. Even more generally, the output object has to be an $\mathbf{EM}(\mathcal{P})$-algebra. Using our lifting of the functor $(-)^A$, another example is $\mathcal{P}(X)^A$ for a finite set X, which allows us to learn nondeterminstic

Mealy machines. However, note that in this model each combination of an origin state and an input symbol determines both the set of next states and the set of outputs; in a common nondeterministic Mealy machine model the set of outputs depends also on the next state. This is the case for the automata learned by El-Fakih et al. [34], Pacharoen et al. [57], and Khalili and Tacchella [48]. Yet another automaton is learned by Volpato and Tretmans [68]. Although it is not directly clear how these automata would fit in our framework, the fact that the algorithms record sets of outputs in their tables suggests that they might benefit from exploiting the algebraic structure of the table analogous to either our definitions or those by Bollig et al. [22].

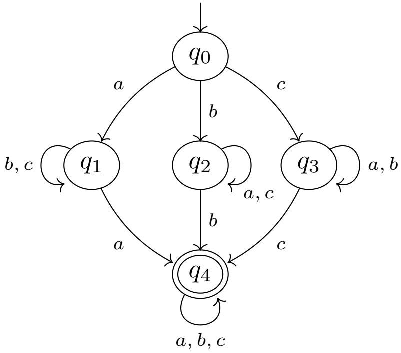


Figure 7.1: DA over $\{a, b, c\}$ displaying a symmetry in its language

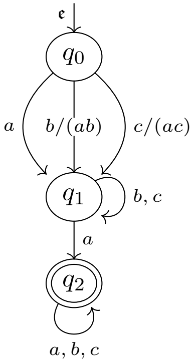


Figure 7.2: T-automaton for Figure 7.1

7 Symmetry

Another compression that may be applied to an automaton is with respect to a certain symmetry. For instance, a symmetry can be observed in Figure 7.1. The states q_1, q_2, and q_3 are equivalent up to a renaming of the alphabet. To describe such relations within the framework developed in the previous section, we introduce some basic group theory.

Definition 7.1 (Group)**.** A *group* is a set $\mathfrak{G}$ equipped with an associative binary operation, denoted by juxtaposition, for which there is an *identity element* $\mathfrak{e}$ and an *inverse element* $\mathfrak{g}^{-1}$ for each $\mathfrak{g} \in \mathfrak{G}$. Thus, for $\mathfrak{g}, \mathfrak{h}, \mathfrak{i} \in \mathfrak{G}$,

$$(\mathfrak{g}\mathfrak{h})\mathfrak{i} = \mathfrak{g}(\mathfrak{h}\mathfrak{i}) \qquad \mathfrak{e}\mathfrak{g} = \mathfrak{g} = \mathfrak{g}\mathfrak{e} \qquad \mathfrak{g}\mathfrak{g}^{-1} = \mathfrak{e} = \mathfrak{g}^{-1}\mathfrak{g}.$$

Lemma 7.2. *In any group $\mathfrak{G}$, we have*

$$\mathfrak{e}^{-1} = \mathfrak{e} \qquad\qquad (\mathfrak{g}_1\mathfrak{g}_2)^{-1} = \mathfrak{g}_2^{-1}\mathfrak{g}_1^{-1}.$$

Proof. We just need to realize that inverse elements are unique: if $\mathfrak{g}\mathfrak{h} = \mathfrak{e}$, then $\mathfrak{h} = \mathfrak{g}^{-1}\mathfrak{g}\mathfrak{h} = \mathfrak{g}^{-1}$. Thus, the desired results follow from

$$\mathfrak{e}\mathfrak{e} = \mathfrak{e} \qquad\qquad \mathfrak{g}_1\mathfrak{g}_2\mathfrak{g}_2^{-1}\mathfrak{g}_1^{-1} = \mathfrak{g}_1\mathfrak{g}_1^{-1} = \mathfrak{e}. \qquad \square$$

Our main example of a group is the permutation group $\mathsf{Perm}(\mathcal{V})$ on a set $\mathcal{V}$. It is defined as the set of permutations of $\mathcal{V}$ (bijections $\mathcal{V} \to \mathcal{V}$), with composition as the group operation. We already know that function composition is associative and that identity functions act as identity elements with respect to composition. Clearly, the composition of two permutations is a permutation, and the identity on $\mathcal{V}$ is also a permutation. The inverse of an element of this group is just the inverse function associated with the bijection.

We fix a group $\mathfrak{G}$, which induces a monad $(\mathfrak{G} \times (-), \eta, \mu)$ as follows:

$$\begin{array}{cc} \eta_X \colon X \to \mathfrak{G} \times X & \mu_X \colon \mathfrak{G} \times (\mathfrak{G} \times X) \to \mathfrak{G} \times X \\ \eta_X(x) = (\mathfrak{e}, x) & \mu_X(\mathfrak{g}_1, (\mathfrak{g}_2, x)) = (\mathfrak{g}_1\mathfrak{g}_2, x). \end{array}$$

Naturality of these is left as an exercise, and we only mention that the monad laws correspond precisely to the associativity and identity equations. Note that if some function f is surjective, then so is $\mathsf{id}_{\mathfrak{G}} \times f$—the monad thus preserves our class $\mathcal{E}$ as defined for **Sets**.

An algebra (U, u) for the monad $\mathfrak{G} \times (-)$ is precisely a *left group action.*

Definition 7.3 (Group Action)**.** A *(left) group action* for the group $\mathfrak{G}$ is a set U with a binary operation $\mathfrak{G} \times U \to U$, denoted by a dot, that satisfies

$$\mathfrak{e} \,.\, u = u \qquad\qquad (\mathfrak{g}_1\mathfrak{g}_2) \,.\, u = \mathfrak{g}_1 \,.\, (\mathfrak{g}_2 \,.\, u).$$

If such an operation is provided, U is called a $\mathfrak{G}$*-set.*

Any set X can be made into a $\mathfrak{G}$-set by equipping it with the *trivial action* defined by $\mathfrak{g} \,.\, x = x$ for all $\mathfrak{g} \in \mathfrak{G}$ and $x \in X$. We simply have $\mathfrak{e} \,.\, x = x$ and

$$(\mathfrak{g}_1\mathfrak{g}_2) \,.\, x = x = \mathfrak{g}_1 \,.\, x = \mathfrak{g}_1 \,.\, (\mathfrak{g}_2 \,.\, x).$$

The *free $\mathfrak{G}$-set* on a set U is the set $\mathfrak{G} \times U$ with an action given by applying the group operation: $\mathfrak{g} \,.\, (\mathfrak{h}, u) = (\mathfrak{g}\mathfrak{h}, u)$. The free $\mathfrak{G}$-set is precisely the free $(\mathfrak{G} \times (-))$-algebra.

Given any two $\mathfrak{G}$-sets U and V, an action on $U \times V$ is given by $\mathfrak{g} \,.\, (u, v) = (\mathfrak{g} \,.\, u, \mathfrak{g} \,.\, v)$. For a $\mathfrak{G}$-set X, we define an action on $\mathcal{P}(X)$ as follows. For each $\mathfrak{g} \in \mathfrak{G}$ and $W \subseteq X$, $\mathfrak{g} \,.\, W = \{\mathfrak{g} \,.\, w \mid w \in W\}$. Note that $\mathfrak{e} \,.\, W = W$ and

$$\begin{aligned} \mathfrak{g}_1\mathfrak{g}_2 \,.\, W &= \{\mathfrak{g}_1\mathfrak{g}_2 \,.\, w \mid w \in W\} = \{\mathfrak{g}_1 \,.\, (\mathfrak{g}_2 \,.\, w) \mid w \in W\} \\ &= \mathfrak{g}_1 \,.\, \{\mathfrak{g}_2 \,.\, w \mid w \in W\} = \mathfrak{g}_1 \,.\, (\mathfrak{g}_2 \,.\, W). \end{aligned}$$

For the group $\mathsf{Perm}(\mathcal{V})$, the set $\mathcal{V}$ is a $\mathfrak{G}$-set with an action defined by application: $\mathfrak{g} \,.\, v = \mathfrak{g}(v)$. Indeed, $\mathfrak{e} \,.\, v = \mathfrak{e}(v) = \mathsf{id}_{\mathcal{V}}(v) = v$ and

$$(\mathfrak{g}_1\mathfrak{g}_2) \,.\, v = (\mathfrak{g}_1 \circ \mathfrak{g}_2)(v) = \mathfrak{g}_1(\mathfrak{g}_2(v)) = \mathfrak{g}_1 \,.\, \mathfrak{g}_2(v) = \mathfrak{g}_1 \,.\, (\mathfrak{g}_2 \,.\, v).$$

The algebra homomorphisms for the monad $\mathfrak{G} \times (-)$ are called *equivariant functions*. That is, a function $f \colon U \to V$ between $\mathfrak{G}$-sets U and V is equivariant if and only if $\mathfrak{g} \,.\, f(u) = f(\mathfrak{g} \,.\, u)$ for all $\mathfrak{g} \in \mathfrak{G}$ and $u \in U$.

Given an equivariant function $f \colon \mathfrak{G} \times U \to V$ for a set U and a $\mathfrak{G}$-set V, the function $f^\dagger \colon U \to V$ is given by $f^\dagger(u) = f(\mathfrak{e}, v)$. Given a function $g \colon U \to V$, the equivariant function $g^\sharp \colon \mathfrak{G} \times U \to V$ is given by $g^\sharp(\mathfrak{g}, u) = \mathfrak{g} \,.\, g(u)$.

In order to lift the functor $(-)^A$ to the category of actions for $\mathfrak{G}$, we assume an action defined on the alphabet A. If no such action is supplied, we can always take it to be the trivial action.

Proposition 7.4. *The collection of components*

$$\begin{gathered} \rho_X \colon \mathfrak{G} \times X^A \to (\mathfrak{G} \times X)^A \\ \rho_X(\mathfrak{g}, f)(a) = (\mathfrak{g}, f(\mathfrak{g}^{-1} \,.\, a)) \end{gathered}$$

defines a distributive law.

Proof. To see that ρ is a natural transformation, consider any function $f\colon U \to V$. Showing commutativity of

$$\begin{tikzcd}
\mathfrak{G} \times U^A \arrow[r, "\rho"] \arrow[d, "\mathsf{id}\times f^A"'] & (\mathfrak{G} \times U)^A \arrow[d, "(\mathsf{id}\times f)^A"] \\
\mathfrak{G} \times V^A \arrow[r, "\rho"] & (\mathfrak{G} \times V)^A
\end{tikzcd}$$

amounts to a simple calculation:

$$\begin{aligned}
&((\mathsf{id} \circ f)^A \circ \rho)(\mathfrak{g}, h)(a) \\
={}& ((\mathsf{id} \circ f)^A(\rho(\mathfrak{g}, h)))(a) \\
={}& ((\mathsf{id} \circ f)(\rho(\mathfrak{g}, h)(a)) && \text{(definition of } (-)^A\text{)} \\
={}& (\mathsf{id} \circ f)(\mathfrak{g}, h(\mathfrak{g}^{-1} \,.\, a)) && \text{(definition of } \rho\text{)} \\
={}& (\mathfrak{g}, (f \circ h)(\mathfrak{g}^{-1} \,.\, a)) \\
={}& \rho(\mathfrak{g}, f \circ h)(a) && \text{(definition of } \rho\text{)} \\
={}& \rho(\mathfrak{g}, f^A(h))(a) && \text{(definition of } (-)^A\text{)} \\
={}& (\rho \circ (\mathsf{id} \times f^A))(\mathfrak{g}, h)(a).
\end{aligned}$$

Moving on to revealing that ρ is a distributive law, we have

$$\begin{tikzcd}
X^A \arrow[r, "\eta"] \arrow[dr, "\eta^A"'] & \mathfrak{G} \times X^A \arrow[d, "\rho"] \\
& (\mathfrak{G} \times X)^A
\end{tikzcd}$$

$$\begin{aligned}
\eta^A(f)(a) &= \eta(f(a)) \\
&= (\mathfrak{e}, f(a)) \\
&= (\mathfrak{e}, f(\mathfrak{e} \,.\, a)) \\
&= (\mathfrak{e}, f(\mathfrak{e}^{-1} \,.\, a)) \\
&= \rho(\mathfrak{e}, f)(a) \\
&= (\rho \circ \eta)(f)(a)
\end{aligned}$$

and

$$\begin{tikzcd}
\mathfrak{G} \times (\mathfrak{G} \times X^A) \arrow[r, "\mathsf{id}\times\rho"] \arrow[d, "\mu"'] & \mathfrak{G} \times (\mathfrak{G} \times X)^A \arrow[r, "\rho"] & (\mathfrak{G} \times (\mathfrak{G} \times X))^A \arrow[d, "\mu^A"] \\
\mathfrak{G} \times X^A \arrow[rr, "\rho"] & & (\mathfrak{G} \times X)^A
\end{tikzcd}$$

$$\begin{aligned}
&(\mu^A \circ \rho \circ (\mathsf{id} \times \rho))(\mathfrak{g}_1, (\mathfrak{g}_2, f))(a) && \\
&= (\mu^A \circ \rho)(\mathfrak{g}_1, \rho(\mathfrak{g}_2, f))(a) && \\
&= (\mu \circ \rho(\mathfrak{g}_1, \rho(\mathfrak{g}_2, f)))(a) && \text{(definition of } (-)^A\text{)} \\
&= \mu(\mathfrak{g}_1, \rho(\mathfrak{g}_2, f)(\mathfrak{g}_1^{-1} \,.\, a)) && \text{(definition of } \rho\text{)} \\
&= \mu(\mathfrak{g}_1, (\mathfrak{g}_2, f(\mathfrak{g}_2^{-1} \,.\, \mathfrak{g}_1^{-1} \,.\, a))) && \text{(definition of } \rho\text{)} \\
&= (\mathfrak{g}_1\mathfrak{g}_2, f(\mathfrak{g}_2^{-1} \,.\, \mathfrak{g}_1^{-1} \,.\, a)) && \text{(definition of } \mu\text{)} \\
&= (\mathfrak{g}_1\mathfrak{g}_2, f(\mathfrak{g}_2^{-1}\mathfrak{g}_1^{-1} \,.\, a)) && \\
&= (\mathfrak{g}_1\mathfrak{g}_2, f((\mathfrak{g}_1\mathfrak{g}_2)^{-1} \,.\, a)) && \\
&= \rho(\mathfrak{g}_1\mathfrak{g}_2, f)(a) && \text{(definition of } \rho\text{)} \\
&= (\rho \circ \mu)(\mathfrak{g}_1, (\mathfrak{g}_2, f))(a) && \text{(definition of } \mu\text{).} \qquad \square
\end{aligned}$$

The resulting lifting of $(-)^A\colon \mathbf{Sets} \to \mathbf{Sets}$ to $\mathbf{EM}(\mathfrak{G} \times (-))$ works as follows. Given a $\mathfrak{G}$-set U, the set U^A becomes a $\mathfrak{G}$-set with the action given by $(\mathfrak{g} \,.\, f)(a) = (\mathfrak{g} \,.\, f)(\mathfrak{g}^{-1} \,.\, a)$ for all $g \in \mathfrak{G}$, $f \in U^A$, and $a \in A$.

Let us assume any action on the output object Y. In examples with $Y = 2$ we will assume the trivial action.

We recover the following semantics of T-automata in this setting. Apart from the current state, one keeps track of a group element. The initial state thus consists of a state and a group element. If the current group element is $\mathfrak{g}$, and an output y is defined for the current state, the actual output is $\mathfrak{g} \,.\, y$. On reading an input symbol a, we find the transition corresponding to the current state and input symbol $\mathfrak{g}^{-1} \,.\, a$, which yields the next state and a group element $\mathfrak{h}$. The next group element then becomes $\mathfrak{g}\mathfrak{h}$.

An example for the group $\mathsf{Perm}(\{a, b, c\})$ is depicted in Figure 7.2. The initial state arrow is labeled with the initial group element. An arrow labeled by $a/\mathfrak{h}$ indicates a transition on input symbol a with a resulting group element $\mathfrak{h}$; if only an input symbol is given, the intended group element is $\mathfrak{e}$. The group element denoted by (ab) is the permutation that swaps a and b while leaving all other symbols intact.

The action on the alphabet A may be extended to an action on words in A^* as follows:

$$\mathfrak{g} \,.\, \varepsilon = \varepsilon \qquad\qquad \mathfrak{g} \,.\, av = (\mathfrak{g} \,.\, a)(\mathfrak{g} \,.\, v).$$

We can read from the proof of Proposition 6.12 and the definition of observability maps in **Sets** that the action $\omega \colon \mathfrak{G} \times Y^{A^*} \to Y^{A^*}$ is given by

$$\omega(\mathfrak{g}, l)(\varepsilon) = \mathfrak{g} \,.\, l(\varepsilon) \qquad \omega(\mathfrak{g}, l)(av) = \omega(\mathfrak{g}, \delta_\Omega(l)(\mathfrak{g}^{-1} \,.\, a))(v),$$

which by a simple inductive argument involving the definition of δ_Ω can be written as

$$\omega(\mathfrak{g}, l)(v) = \mathfrak{g} \,.\, l(\mathfrak{g}^{-1} \,.\, v).$$

Thus,

$$(\mathfrak{g} \,.\, l)(v) = \mathfrak{g} \,.\, l(\mathfrak{g}^{-1} \,.\, v).$$

A language is now an equivariant function $\mathcal{L} \colon \mathfrak{G} \times A^* \to Y$. By Proposition 6.7 it can be seen as a function $\mathcal{L}^\dagger \colon A^* \to Y$.

Proposition 7.5. *The total response $t_\mathcal{L} \colon \mathfrak{G} \times A^* \to Y^{A^*}$ is given by*

$$t_\mathcal{L}(\mathfrak{g}, u)(v) = \mathcal{L}(\mathfrak{g}, u \cdot (\mathfrak{g}^{-1} \,.\, v)).$$

Proof. Using Proposition 6.13, we have

$$\begin{aligned} t_\mathcal{L}(\mathfrak{g}, u)(v) = t^\sharp_{\mathcal{L}^\dagger}(\mathfrak{g}, u)(v) &= (\mathfrak{g} \,.\, t_{\mathcal{L}^\dagger}(u))(v) \\ &= \mathfrak{g} \,.\, t_{\mathcal{L}^\dagger}(u)(\mathfrak{g}^{-1} \,.\, v) \\ &= \mathfrak{g} \,.\, \mathcal{L}^\dagger(u \cdot (\mathfrak{g}^{-1} \,.\, v)) \\ &= \mathfrak{g} \,.\, \mathcal{L}(\mathfrak{e}, u \cdot (\mathfrak{g}^{-1} \,.\, v)) \\ &= \mathcal{L}(\mathfrak{g}, u \cdot (\mathfrak{g}^{-1} \,.\, v)). \end{aligned}$$

□

Proposition 7.6. *The alternative language representation is given by*

$$\overline{\mathcal{L}} \colon \mathfrak{G} \times 1 \to Y^{A^*} \qquad\qquad \overline{\mathcal{L}}(\mathfrak{g}, *)(v) = \mathcal{L}(\mathfrak{g}, \mathfrak{g}^{-1} \,.\, v).$$

Proof. We read from Proposition 6.14 that

$$\begin{aligned}\overline{\mathcal{L}}(\mathfrak{g}, *)(v) = \overline{\mathcal{L}^\dagger}^\sharp(\mathfrak{g}, *)(v) &= (\mathfrak{g} \,.\, \overline{\mathcal{L}^\dagger}(*))(v) \\ &= \mathfrak{g} \,.\, \overline{\mathcal{L}^\dagger}(*)(\mathfrak{g}^{-1} \,.\, v) \\ &= \mathfrak{g} \,.\, \mathcal{L}^\dagger(\mathfrak{g}^{-1} \,.\, v) \\ &= \mathfrak{g} \,.\, \mathcal{L}(\mathfrak{e}, \mathfrak{g}^{-1} \,.\, v) \\ &= \mathcal{L}(\mathfrak{g}, \mathfrak{g}^{-1} \,.\, v). \qquad \square\end{aligned}$$

7.1 Observation Tables

Observation tables in general do not lift as nicely to this setting as they did to $\mathbf{EM}(\mathcal{P})$. For example, consider $A = \{a, b, c\}$ and $\mathfrak{G} = \mathsf{Perm}(A)$, where the action on A is defined by application. The set 2 is equipped with the trivial action. Suppose there is an action on $2^{\{a\}}$ such that $\pi_{\{a\}} \colon 2^{A^*} \to 2^{\{a\}}$ is equivariant. Let $l, l' \in 2^{A^*}$ be the languages $\{b\}$ and $\{c\}$, respectively. Then we must have

$$\pi_{\{a\}}((ab) \,.\, l)(a) = 1 \neq 0 = \pi_{\{a\}}((ab) \,.\, l')(a),$$

but by equivariance,

$$\pi_{\{a\}}((ab) \,.\, l) = (ab) \,.\, \pi_{\{a\}}(l) = (ab) \,.\, \pi_{\{a\}}(l') = \pi_{\{a\}}((ab) \,.\, l').$$

As a workaround, we redefine an observation table approximation for this setting as follows:

$$\widehat{\mathsf{T}}(S, E) = (\mathfrak{G} \times S, Y^{\widehat{E}}, \mathsf{id}_{\mathfrak{G}} \times \sigma_S, \pi_{\widehat{E}}) \qquad \widehat{E} = \{g \,.\, e \mid e \in E\}.$$

It remains to show that $Y^{\widehat{E}}$ is a $\mathfrak{G}$-set in such a way that $\pi_{\widehat{E}}$ is equivariant. Given $l \in Y^{\widehat{E}}$, we define

$$(\mathfrak{g} \,.\, l)(\hat{e}) = \mathfrak{g} \,.\, l(\mathfrak{g}^{-1} \,.\, \hat{e}),$$

which satisfies

$$\begin{aligned} (\mathfrak{e}\,.\,l)(\hat{e}) &= \mathfrak{e}\,.\,l(\mathfrak{e}^{-1}\,.\,\hat{e}) & (\mathfrak{g}_1\cdot(\mathfrak{g}_2\,.\,l))(\hat{e}) &= \mathfrak{g}_1\cdot(\mathfrak{g}_2\,.\,l)(\mathfrak{g}_1^{-1}\,.\,\hat{e}) \\ &= l(\mathfrak{e}^{-1}\,.\,\hat{e}) & &= \mathfrak{g}_1\cdot(\mathfrak{g}_2\,.\,l(\mathfrak{g}_2^{-1}\cdot(\mathfrak{g}_1^{-1}\,.\,\hat{e}))) \\ &= l(\mathfrak{e}\,.\,\hat{e}) & &= \mathfrak{g}_1\mathfrak{g}_2\,.\,l(\mathfrak{g}_2^{-1}\mathfrak{g}_1^{-1}\,.\,\hat{e}) \\ &= l(\hat{e}) & &= \mathfrak{g}_1\mathfrak{g}_2\,.\,l((\mathfrak{g}_1\mathfrak{g}_2)^{-1}\,.\,\hat{e}) \\ & & &= (\mathfrak{g}_1\mathfrak{g}_2\,.\,l)(\hat{e}). \end{aligned}$$

Furthermore, for $l \in Y^{A^*}$,

$$\begin{aligned} (\mathfrak{g}\,.\,\pi_{\widehat{E}}(l))(\hat{e}) = \mathfrak{g}\,.\,\pi_{\widehat{E}}(l)(\mathfrak{g}^{-1}\,.\,\hat{e}) = \mathfrak{g}\,.\,l(\mathfrak{g}^{-1}\,.\,\hat{e}) &= (\mathfrak{g}\,.\,l)(\hat{e}) \\ &= \pi_{\widehat{E}}(\mathfrak{g}\,.\,l)(\hat{e}). \end{aligned}$$

Proposition 7.7. *The observation structure associated with the observation table approximation* $\widehat{\mathsf{T}}(S, E)$ *is given by*

$$\begin{aligned} \xi\colon \mathfrak{G}\times S &\to Y^{\widehat{E}} & \xi(\mathfrak{g}, s)(\hat{e}) &= \mathcal{L}(\mathfrak{g}, s\cdot(\mathfrak{g}^{-1}\,.\,\hat{e})) \\ \xi_I\colon \mathfrak{G}\times 1 &\to Y^{\widehat{E}} & \xi_I(\mathfrak{g}, *)(\hat{e}) &= \mathcal{L}(\mathfrak{g}, \mathfrak{g}^{-1}\,.\,\hat{e}) \\ \xi_Y\colon \mathfrak{G}\times S &\to Y & \xi_Y(\mathfrak{g}, s) &= \mathcal{L}(\mathfrak{g}, s) \\ \xi_\delta\colon \mathfrak{G}\times S &\to Y^{\widehat{E}} & \xi_\delta(\mathfrak{g}, s)(a)(\hat{e}) &= \mathcal{L}(\mathfrak{g}, s\cdot(\mathfrak{g}^{-1}\,.\,a\hat{e})). \end{aligned}$$

Proof. Apply Proposition 6.23. The wrapper in **B** that shares its observation structure with $\widehat{\mathsf{T}}(S, E)$ is the one associated with the approximation $\mathsf{T}(S, \widehat{E})$. The given definitions follow directly after writing down in each case the definition of $(-)^\sharp$ and the relevant action. □

Recall that the potential hypothesis H arises as the factorization of ξ:

$$\begin{tikzcd} \mathfrak{G}\times S \arrow[r, two heads, "e"] \arrow[rr, bend right, "\xi"'] & H \arrow[r, tail, "m"] & Y^{\widehat{E}} \end{tikzcd}$$

Proposition 7.8. *An approximation* $\widehat{\mathsf{T}}(S,E)$ *is initialized if and only if there exist* $\mathfrak{g} \in \mathfrak{G}$ *and* $s \in S$ *such that* $\xi(\mathfrak{g}, s) = \xi_I^\dagger(*)$.

Proof. We know through Lemma 2.17 and Lemma 6.15 that $\widehat{\mathsf{T}}(S,E)$ is initialized if and only if $\mathsf{im}(\xi_I) \subseteq \mathsf{im}(m)$. Since $\mathsf{im}(m) = \mathsf{im}(\xi)$, this translates to the property that for all $\mathfrak{h} \in \mathfrak{G}$ there exist $\mathfrak{i} \in \mathfrak{G}$ and $s \in S$ such that $\xi(\mathfrak{i}, s) = \xi_I(\mathfrak{h}, *)$. Note that if there is a $\mathfrak{g}$ such that $\xi(\mathfrak{g}, s) = \xi_I^\dagger(*) = \xi_I(\mathfrak{e}, *)$, then

$$\xi(\mathfrak{h}\mathfrak{g}, s) = \mathfrak{h} \,.\, \xi(\mathfrak{g}, s) = \mathfrak{h} \,.\, \xi_I(\mathfrak{e}, *) = \xi_I(\mathfrak{h}, *),$$

and the converse certainly holds. □

Proposition 7.9. *An approximation* $\widehat{\mathsf{T}}(S,E)$ *is responsive if and only if for all* $\mathfrak{g} \in \mathfrak{G}$ *and* $s_1, s_2 \in S$ *such that* $\xi(\mathfrak{g}, s_1) = \xi^\dagger(s_2)$ *we have* $\mathcal{L}(\mathfrak{g}, s_1) = \mathcal{L}^\dagger(s_2)$.

Proof. Using Lemma 2.17 and Lemma 6.15, we have that $\widehat{\mathsf{T}}(S,E)$ is responsive if and only if $\mathsf{ker}(e) \subseteq \mathsf{ker}(\xi_Y)$. Note that because m is injective, we have $\mathsf{ker}(e) = \mathsf{ker}(m \circ e) = \mathsf{ker}(\xi)$.

We want to prove the following properties equivalent:

1. for all $\mathfrak{g} \in \mathfrak{G}$ and $s_1, s_2 \in S$, if $\xi(\mathfrak{g}, s_1) = \xi(\mathfrak{e}, s_2)$, then also $\xi_Y(\mathfrak{g}, s_1) = \xi_Y(\mathfrak{e}, s_2)$
2. for all $\mathfrak{g}_1, \mathfrak{g}_2 \in \mathfrak{G}$ and $s_1, s_2 \in S$, if $\xi(\mathfrak{g}_1, s_1) = \xi(\mathfrak{g}_2, s_2)$, then also $\xi_Y(\mathfrak{g}_1, s_1) = \xi_Y(\mathfrak{g}_2, s_2)$

Clearly, the second implies the first. Assume the first and consider $\mathfrak{g}_1, \mathfrak{g}_2 \in \mathfrak{G}$ and $s_1, s_2 \in S$ such that $\xi(\mathfrak{g}_1, s_1) = \xi(\mathfrak{g}_2, s_2)$. Then because

$$\xi(\mathfrak{g}_2^{-1}\mathfrak{g}_1, s_1) = \mathfrak{g}_2^{-1} \,.\, \xi(\mathfrak{g}_1, s_1) = \mathfrak{g}_2^{-1} \,.\, \xi(\mathfrak{g}_2, s_2) = \xi(\mathfrak{e}, s_2),$$

we have

$$\xi_Y(\mathfrak{g}_2^{-1}\mathfrak{g}_1, s_1) = \xi_Y(\mathfrak{e}, s_2)$$

and so

$$\xi_Y(\mathfrak{g}_1, s_1) = \mathfrak{g}_2 \,.\, \xi_Y(\mathfrak{g}_2^{-1}\mathfrak{g}_1, s_1) = \mathfrak{g}_2 \,.\, \xi_Y(\mathfrak{e}, s_2) = \xi_Y(\mathfrak{g}_2, s_2).$$ □

Proposition 7.10. *An approximation* $\widehat{\mathsf{T}}(S,E)$ *is closed if and only if for all* $s \in S$ *and* $a \in A$ *there exist* $\mathfrak{g} \in \mathfrak{G}$ *and* $s' \in S$ *such that* $\xi(\mathfrak{g}, s') = \xi^{\dagger}_{\delta}(s)(a)$.

Proof. Using Corollary 6.25, the proof is only a slight variation on the proof of Proposition 3.12. Specifically, the domain of the function e is here $\mathfrak{G} \times S$ rather than just S. □

Proposition 7.11. *An approximation* $\widehat{\mathsf{T}}(S,E)$ *is consistent if and only if for all* $\mathfrak{g} \in \mathfrak{G}$ *and* $s_1, s_2 \in S$ *such that* $\xi(\mathfrak{g}, s_1) = \xi^{\dagger}(s_2)$ *we have* $\xi_{\delta}(\mathfrak{g}, s_1)(a) = \xi^{\dagger}_{\delta}(s_2)(a)$ *for each* $a \in A$.

Proof. The proof is completely analogous to the proof of Proposition 7.9, with ξ_{δ} substituted for ξ_Y. □

Consider $\mathfrak{G} = \mathsf{Perm}(\{a,b,c\})$. Table 7.3a shows an observation table with $S = \{\varepsilon\}$ and $E = \{\varepsilon, a\}$. (Note that $\widehat{E} = \{\varepsilon, a, b, c\}$.) This table is not closed: there is no permutation of $\xi^{\dagger}(\varepsilon)$ by which we obtain $\xi^{\dagger}_{\delta}(\varepsilon)(a)$. Table 7.3b, on the other hand, *is* closed: we have $\xi^{\dagger}_{\delta}(\varepsilon)(b) = \xi((ab), a)$, $\xi^{\dagger}_{\delta}(\varepsilon)(c) = \xi((ac), a)$, and the other lower rows are simply included in the upper part.

7.2 Infinite Alphabets

One may contemplate a generalization of Figure 7.2 over an infinite alphabet, i.e., an automaton accepting all words that begin and end with the same symbol drawn from an infinite set. It is not hard to imagine that such an automaton could be described by finite means. For instance, we note that for all pairs of states in Figure 7.2 the transitions between those states are in some sense almost symmetrical. The transitions from q_0 to q_1 are all of the form $x/(ax)$ for $x \in \mathcal{V}$ ($(aa) = \mathfrak{e}$); the transitions from q_1 to q_1 are mostly of the form $x/\mathfrak{e}$, with the exception of $x = a$. For the generalization to an infinite alphabet it is vital that there are only finitely many such exceptions.

	ε	a	b	c
ε	0	0	0	0
a	0	1	0	0
b	0	0	1	0
c	0	0	0	1

(a) Example table

	ε	a	b	c
ε	0	0	0	0
a	0	1	0	0
aa	1	1	1	1
b	0	0	1	0
c	0	0	0	1
ab	0	1	0	0
ac	0	1	0	0
aaa	1	1	1	1
aab	1	1	1	1
aac	1	1	1	1

(b) Dynamical table

Table 7.3: Observation tables for the language described in Figure 7.1

Let us now assume that $\mathcal{V}$ is an infinite set. To make precise the assumption that the language to be learned is almost symmetrical, we need the notion of a *support.*

Definition 7.12 (Support)**.** An element x in a $\mathfrak{G}$-set X is *supported by* $C \subseteq \mathcal{V}$ if for all $\mathfrak{g} \in \mathfrak{G}$ such that $\mathfrak{g} \,.\, c = c$ for all $c \in C$ we have $\mathfrak{g} \,.\, x = x$.

Recall that the action on C is just application of the permutation: $\mathfrak{g} \,.\, c = \mathfrak{g}(c)$. Every element x of a $\mathfrak{G}$-set is supported by the whole of $\mathcal{V}$: if $\mathfrak{g} \,.\, c = c$ for each $c \in \mathcal{V}$, then $\mathfrak{g}$ must be the identity $\mathfrak{e}$ on $\mathcal{V}$, and this ensures $\mathfrak{g} \,.\, x = x$. Every $v \in \mathcal{V}$ is trivially supported by $\{v\}$. If X and W are $\mathfrak{G}$-sets, $x \in X$ is supported by $C \subseteq \mathcal{V}$, and $w \in W$ is supported by $D \subseteq \mathcal{V}$, then (x, w) is supported by $C \cup D$. Similarly, a word over an alphabet is supported by the union of any choice of supports for the individual symbols in that word. If an element x in a $\mathfrak{G}$-set is supported by a set $C \subseteq \mathcal{V}$, then for each $\mathfrak{g} \in \mathfrak{G}$, $\mathfrak{g} \,.\, x$ is supported by $\mathfrak{g} \,.\, C$.

We assume that each finitely supported element x of a $\mathfrak{G}$-set has a *least* support, i.e., a set $C_x \subseteq \mathcal{V}$ that supports x and that is included in every subset of $\mathcal{V}$ supporting x. This will play a key role in determining dynamism of observation table approximations.

The abstract theory in this section works for any *subgroup* $\mathfrak{G}$ of $\mathsf{Perm}(\mathcal{V})$ (subject to the assumptions that accumulate throughout). A *subgroup* $\mathfrak{G}$ of $\mathsf{Perm}(\mathcal{V})$ is a set $\mathfrak{G} \subseteq \mathsf{Perm}(\mathcal{V})$ that forms a group by inheriting the operation from $\mathsf{Perm}(\mathcal{V})$. Our main example is the case that $\mathfrak{G} = \mathsf{Perm}(\mathcal{V})$, where $\mathcal{V}$ is *countably* infinite. This is referred to as the *equality symmetry.* Gabbay and Pitts proved that the equality symmetry admits least supports [35, Proposition 3.4].

We may now pose the following formal restriction on the language to be learned: we assume that $t^{\dagger}_{\mathcal{L}}(\varepsilon)$ has finite support. The set $C_{t^{\dagger}_{\mathcal{L}}(\varepsilon)}$ can be seen as a set of "special values" for which a realization of the language may have asymmetrical behavior in any state. We assume that some finite superset of this set is known in advance to the learner.

A common problem is to determine, given a $\mathfrak{G}$-set X and $x, y \in X$, whether there exists a $\mathfrak{g} \in \mathfrak{G}$ such that $\mathfrak{g} \,.\, x = y$. Most importantly, this problem occurs for rows x and y when determining whether the approximation is dynamical. When $\mathfrak{G}$ was finite, we could simply try all $\mathfrak{g} \in \mathfrak{G}$, but at present we need to reduce the search space drastically.

To find $\mathfrak{g}$ such that $\mathfrak{g} \,.\, x = y$ for finitely supported x and y, it turns out to be sufficient to consider only $\mathfrak{g}$ satisfying $\mathfrak{g} \,.\, C_x = C_y$. Equivalently, this result states that the function $C_{(-)} \colon X \to \mathcal{P}X$ is equivariant [60, Proposition 2.11].

Proposition 7.13. *Let x and y be finitely supported elements from the same $\mathfrak{G}$-set. If $\mathfrak{g} \in \mathfrak{G}$ is such that $\mathfrak{g} \,.\, x = y$, then $\mathfrak{g} \,.\, C_x = C_y$.*

Proof. We first show that y is supported by $\mathfrak{g} \,.\, C_x$. Suppose $\mathfrak{h} \in \mathfrak{G}$ is such that for each $c \in \mathfrak{g} \,.\, C_x$ we have $\mathfrak{h} \,.\, c = c$. It follows that for each $c \in C_x$ we have $\mathfrak{h}\mathfrak{g} \,.\, c = \mathfrak{g} \,.\, c$, which gives $\mathfrak{g}^{-1}\mathfrak{h}\mathfrak{g} \,.\, c = c$. Because

C_x supports x, this implies $\mathfrak{g}^{-1}\mathfrak{h}\mathfrak{g} \mathbin{.} x = x$, from which we deduce that

$$\mathfrak{h} \mathbin{.} y = \mathfrak{h}\mathfrak{g} \mathbin{.} x = \mathfrak{g}\mathfrak{g}^{-1}\mathfrak{h}\mathfrak{g} \mathbin{.} x = \mathfrak{g} \mathbin{.} x = y.$$

Thus, $C_y \subseteq \mathfrak{g} \mathbin{.} C_x$.

To see that $\mathfrak{g} \mathbin{.} C_x$ is the least support of y, note that since $\mathfrak{g}^{-1} \mathbin{.} y = x$, we can use the first part of the proof to find that $C_x \subseteq \mathfrak{g}^{-1} \mathbin{.} C_y$. Then

$$\mathfrak{g} \mathbin{.} C_x \subseteq \mathfrak{g} \mathbin{.} \mathfrak{g}^{-1} \mathbin{.} C_y = \mathfrak{g}\mathfrak{g}^{-1} \mathbin{.} C_y = \mathfrak{e} \mathbin{.} C_y = C_y,$$

so we conclude that $\mathfrak{g} \mathbin{.} C_x = C_y$. □

Note that if $\mathfrak{g}, \mathfrak{h} \in \mathfrak{G}$ are such that $\mathfrak{g} \mathbin{.} c = \mathfrak{h} \mathbin{.} c$ for all $c \in C_x$, then $\mathfrak{h}^{-1}\mathfrak{g} \mathbin{.} c = c$ for all $c \in C_x$. Because C_x supports x this means that $\mathfrak{h}^{-1}\mathfrak{g} \mathbin{.} x = x$, and thus $\mathfrak{g} \mathbin{.} x = \mathfrak{h} \mathbin{.} x$. The number of group elements that are to be considered can be bounded by the finite number of bijections $C_x \to C_y$. Given such a bijection, we assume to be able to determine whether it extends to a permutation $\mathfrak{g} \in \mathfrak{G}$. If for each of the bijections $C_x \to C_y$ the associated $\mathfrak{g}$ does not satisfy $\mathfrak{g} \mathbin{.} x = y$, then there is no $\mathfrak{g} \in \mathfrak{G}$ such that $\mathfrak{g} \mathbin{.} x = y$.

For the equality symmetry, the desired extensions can be formed as follows.

Proposition 7.14. *For each bijection $f \colon C \to D$ between finite sets $C, D \subseteq \mathcal{V}$ there exists $\mathfrak{g}_f \in \mathsf{Perm}(\mathcal{V})$ such that $\mathfrak{g}_f(c) = f(c)$ for all $c \in C$.*

Proof. We prove by induction on $n \in \mathbb{N}$ that for all bijections $f \colon C \to D$ for finite sets C with $|C| = n$ there exists $\mathfrak{g}_f \in \mathsf{Perm}(\mathcal{V})$ such that $\mathfrak{g}_f(c) = \check{f}(c)$ for all $c \in C$. If $n = 0$, then clearly $\mathfrak{g}_f = \mathfrak{e}$ suffices regardless of f.

Suppose $n = m + 1$ for some $m \in \mathbb{N}$. We write

$$C = \{c_1, c_2, \ldots, c_n\}.$$

Define $\check{f} \colon C \setminus \{c_n\} \to D \setminus \{f(c_n)\}$ by $\check{f}(c) = f(c)$ for $c \in C \setminus \{c_n\}$. By the induction hypothesis we can find $\mathfrak{g}_{\check{f}} \in \mathsf{Perm}(\mathcal{V})$ such that

$\mathfrak{g}_{\check{f}}(c) = f(c)$ for all $c \in C \setminus \{c_n\}$. Define

$$x = \mathfrak{g}_{\check{f}}^{-1}(f(c_n)) \qquad\qquad \mathfrak{g}_f = \mathfrak{g}_{\check{f}} \circ (x c_n).$$

Note that $x \notin C \setminus \{c_n\}$, for otherwise

$$f(c_n) = \mathfrak{g}_{\check{f}}(x) = \check{f}(x) \in D \setminus \{f(c_n)\}.$$

Hence, for each $c \in C$ we have either $c = c_n$, which means that $\mathfrak{g}_f(c) = \mathfrak{g}_{\check{f}}(x) = f(c_n) = f(c)$; or $\mathfrak{g}_f(c) = \mathfrak{g}_{\check{f}}(c) = \check{f}(c) = f(c)$. □

To compare rows, it remains to find the least support of a row. We establish first a finite support for each row, and subsequently indicate how such a support can be reduced to the least support. This reduction works if the group $\mathfrak{G}$ is *fungible* [21].

Lemma 7.15. *For all $u \in A^*$, $t^\dagger_{\mathcal{L}}(u)$ has finite support. More specifically,*

$$C_{t^\dagger_{\mathcal{L}}(u)} \subseteq C_{t^\dagger_{\mathcal{L}}(\varepsilon)} \cup C_u.$$

Proof. We need to show that $t^\dagger_{\mathcal{L}}(u)$ is supported by $C_{t^\dagger_{\mathcal{L}}(\varepsilon)} \cup C_u$. Suppose $\mathfrak{g} \in \mathfrak{G}$ is such that for all $c \in C_{t^\dagger_{\mathcal{L}}(\varepsilon)} \cup C_u$, $\mathfrak{g} \,.\, c = c$. Because u is supported by C_u, we have $\mathfrak{g} \,.\, u = u$; since $t^\dagger_{\mathcal{L}}(\varepsilon)$ is supported by $C_{t^\dagger_{\mathcal{L}}(\varepsilon)}$, we have $\mathfrak{g} \,.\, t^\dagger_{\mathcal{L}}(\varepsilon) = t^\dagger_{\mathcal{L}}(\varepsilon)$. Letting $\mathfrak{g}^{-1}$ act on both sides of the latter equation, we observe that $t^\dagger_{\mathcal{L}}(\varepsilon) = \mathfrak{g}^{-1} \,.\, t^\dagger_{\mathcal{L}}(\varepsilon)$, or, equivalently, $\mathcal{L}^\dagger(u) = \mathcal{L}(\mathfrak{g}^{-1}, \mathfrak{g} \,.\, u)$ for all $u \in A^*$. Thus, for all $v \in A^*$,

$$\begin{aligned}
(\mathfrak{g} \,.\, t^\dagger_{\mathcal{L}}(u))(v) &= t_{\mathcal{L}}(\mathfrak{g}, u)(v) \\
&= \mathcal{L}(\mathfrak{g}, u \cdot (\mathfrak{g}^{-1} \,.\, v)) \\
&= \mathcal{L}(\mathfrak{e}, \mathfrak{g} \,.\, (u \cdot (\mathfrak{g}^{-1} \,.\, v))) \\
&= \mathcal{L}(\mathfrak{e}, (\mathfrak{g} \,.\, u) \cdot v) \\
&= \mathcal{L}(\mathfrak{e}, uv) \\
&= t^\dagger_{\mathcal{L}}(u)(v).
\end{aligned}$$

□

Lemma 7.16. *For each finitely supported $l \in Y^{A^*}$, $C_{\pi_{\widehat{E}}(l)} \subseteq C_l$.*

Proof. For all $\mathfrak{g} \in \mathfrak{G}$ such that $\mathfrak{g} \,.\, l = l$, we have, by equivariance of $\pi_{\widehat{E}}$,[9]

$$\mathfrak{g} \,.\, \pi_{\widehat{E}}(l) = \pi_{\widehat{E}}(\mathfrak{g} \,.\, l) = \pi_{\widehat{E}}(l).$$ □

Proposition 7.17. *We have*

$$C_{\xi_I^\dagger(*)} \subseteq C_{t_{\mathcal{L}}^\dagger(\varepsilon)}.$$

Furthermore, for each $s \in S$,

$$C_{\xi^\dagger(s)} \subseteq C_{t_{\mathcal{L}}^\dagger(\varepsilon)} \cup C_s,$$

and for every $a \in A$,

$$C_{\xi_\delta^\dagger(s)(a)} \subseteq C_{t_{\mathcal{L}}^\dagger(\varepsilon)} \cup C_{sa}.$$

Proof. For each $\hat{e} \in \widehat{E}$, we find

$$\begin{aligned}
\xi_I^\dagger(*)(\hat{e}) = \mathcal{L}^\dagger(\hat{e}) = t_{\mathcal{L}}^\dagger(\varepsilon)(\hat{e}) = \pi_{\widehat{E}}(t_{\mathcal{L}}^\dagger(\varepsilon))(\hat{e}) \\
\xi^\dagger(s)(\hat{e}) = \mathcal{L}^\dagger(s\hat{e}) = t_{\mathcal{L}}^\dagger(s)(\hat{e}) = \pi_{\widehat{E}}(t_{\mathcal{L}}^\dagger(s))(\hat{e}) \\
\xi_\delta^\dagger(s)(a)(\hat{e}) = \mathcal{L}^\dagger(sa\hat{e}) = t_{\mathcal{L}}^\dagger(sa)(\hat{e}) = \pi_{\widehat{E}}(t_{\mathcal{L}}^\dagger(sa))(\hat{e})
\end{aligned}$$

The results then follow from Lemma 7.15 and Lemma 7.16. □

Proposition 7.18. *Given an element x from a $\mathfrak{G}$-set, a finite set $C \subseteq \mathcal{V}$ supporting x, and $c \in C$, if $\mathfrak{g} \in \mathfrak{G}$ is such that $\mathfrak{g} \,.\, c \notin C$, and for each $d \in C \setminus \{c\}$, $\mathfrak{g} \,.\, d = d$, then $c \in C_x$ if and only if $\mathfrak{g} \,.\, x \neq x$.*

Proof. Assume first that $c \in C_x$, and suppose that we would have $\mathfrak{g} \,.\, x = x$. Applying Proposition 7.13, we obtain $\mathfrak{g} \,.\, C_x = C_x$, so in particular $\mathfrak{g} \,.\, c \in C_x \subseteq C$. However, $\mathfrak{g} \,.\, c \notin C$. From this contradiction we conclude that $\mathfrak{g} \,.\, x \neq x$.

Now assume instead that $\mathfrak{g} \,.\, x \neq x$, and suppose that we would have $c \notin C_x$. Since $\mathfrak{g} \,.\, d = d$ holds for all $d \in C \setminus \{c\}$, this in particular holds for all $d \in C_x$. Because C_x supports x, it follows that $\mathfrak{g} \,.\, x = x$, contradicting our assumption. Hence, $c \in C_x$. □

[9] The proof of course works for any equivariant function [60, Lemma 2.12].

For the equality symmetry, an appropriate $\mathfrak{g}$ for a given $c \in C$ is easily determined. Since C is finite and $\mathcal{V}$ is countably infinite, we can find $v \in \mathcal{V} \setminus C$. Now $(cv) \in \mathfrak{G}$ has the desired property.

The problem of determining the appropriate similarity of rows has been solved, but in general we are still stuck with an infinite number of rows in the lower part of the table, as well as an infinite number of columns. To remedy this situation, we will place some restrictions on the alphabet A.

For each $\mathfrak{G}$-set X and each element $x \in X$, the *orbit* of x is defined as the set

$$\mathfrak{G} \,.\, x = \{\mathfrak{g} \,.\, x \mid \mathfrak{g} \in \mathfrak{G}\}.$$

The set

$$\mathsf{orbits}(X) = \{\mathfrak{G} \,.\, x \mid x \in X\}$$

partitions X. That is, for each $x \in X$ there is precisely one orbit $O \in \mathsf{orbits}(X)$ such that $x \in O$. We write $x \sim y$ if x and $y \in X$ belong to the same orbit. Thus, $x \sim y$ if and only if there exists $\mathfrak{g} \in \mathfrak{G}$ such that $\mathfrak{g} \,.\, x = y$. This is an equivalence relation.

The $\mathfrak{G}$-set X is *orbit-finite* if the set $\mathsf{orbits}(X)$ is finite. For each such set we fix a set of representatives for its orbits, i.e., a finite set $\mathsf{rep}(X)$ such that for each $O \in \mathsf{orbits}(X)$ the set $O \cap \mathsf{rep}(X)$ is a singleton. Hence, for each $x \in X$ there is precisely one $x' \in \mathsf{rep}(X)$ such that $x \sim x'$.

We will assume that our alphabet A is orbit-finite and *nominal.*

Definition 7.19 (Nominal Set [21])**.** A $\mathfrak{G}$-set X is *nominal* if every $x \in X$ has finite support.

Consider for a moment the equality symmetry and define $A = \mathcal{V}$. For any word $ab \in S \subseteq A^*$ formed from $a, b \in A$ with $a \neq b$, the one-symbol successors of ab are aba, abb, and abc for each $c \in \mathcal{V} \setminus \{a, b\}$. Note that if $c, d \in \mathcal{V} \setminus \{a, b\}$, then $abc \sim abd$. Thus, the finite set $\{\mathfrak{e}, (ab), (ac)\} \subseteq \mathfrak{G}$ has the property that for each successor $ab \cdot (\mathfrak{g} \,.\, a)$ of ab there is $\mathfrak{h} \in \{\mathfrak{e}, (ab), (ac)\}$ such that $ab \cdot (\mathfrak{h} \,.\, a) \sim ab \cdot (\mathfrak{g} \,.\, a)$. Determining an appropriate subset of $\mathfrak{G}$ is essentially an operation on supports: we seek permutations that act differently on the support of a relative to the support of ab.

In the more general setting, we assume that for all finite sets $C, D \subseteq \mathcal{V}$ there is a finite set $R_{C|D} \subseteq \mathfrak{G}$ such that for all $\mathfrak{g} \in \mathfrak{G}$ there exist $\mathfrak{h} \in R_{C|D}$ and $\mathfrak{i} \in \mathfrak{G}$ satisfying $\mathfrak{g} \,.\, c = \mathfrak{i}\mathfrak{h} \,.\, c$ for all $c \in C$ and $\mathfrak{i} \,.\, d = d$ for all $d \in D$. Given finitely supported elements x and y from two (possibly different) $\mathfrak{G}$-sets, we write $R_{x|y}$ for $R_{C_x|C_y}$. We further define for any nominal orbit-finite $\mathfrak{G}$-set U and every finitely supported $v \in V$ from any $\mathfrak{G}$-set V,

$$U \dashv v = \{\mathfrak{h} \,.\, u \mid u \in \mathsf{rep}(U), \mathfrak{h} \in R_{u|v}\}.$$

It follows that for each $u \in U$ there is a $u' \in (U \dashv v)$ such that $(u', v) \sim (u, v)$.

For the above example, we may take $R_{a|ab} = \{\mathfrak{e}, (ab), (ac)\}$, provided that $c \in A \setminus \{a, b\}$. If $\mathsf{rep}(A) = \{a\}$, then $A \dashv ab = \{a, b, c\}$.

Consider the equality symmetry and finite $C, D \subseteq \mathcal{V}$. We may define $R_{C|D}$ as follows. Write

$$C = \{c_1, c_2, \ldots, c_{|C|}\}$$

and determine $X = \{x_1, x_2, \ldots, x_{|C|}\} \subseteq \mathcal{V} \setminus D$ with $x_n \neq x_m$ whenever $n \neq m$, which is possible because $\mathcal{V}$ is countably infinite while C and D are finite. More constructively, without loss of generality we can assume $\mathcal{V} = \mathbb{N}$ and define $x_i = \max(D) + i$ for all $1 \leq i \leq |C|$. Define

$$\begin{aligned} R_{C|D} = \{\mathfrak{g}_f \mid {} & f\colon C \to D \cup X \text{ is injective and} \\ & \qquad f(c_i) \in X \iff f(c_i) = x_i \text{ for all } 1 \leq i \leq |C|\}. \end{aligned}$$

We use Proposition 7.14 here by regarding injections $f\colon C \to D \cup X$ as bijections $C \to \mathsf{im}(f)$.

Given $\mathfrak{g} \in \mathfrak{G}$, define $\mathfrak{h} = \mathfrak{g}_h$ and $\mathfrak{i} = \mathfrak{g}_i$, where

$$\begin{aligned} & h\colon C \to D \cup X && i\colon D \cup \mathfrak{h} \,.\, C \to \mathcal{V} \\ & h(c_i) = \begin{cases} \mathfrak{g}(c_i) & \text{if } \mathfrak{g}(c_i) \in D \\ x_i & \text{if } \mathfrak{g}(c_i) \notin D \end{cases} && i(v) = \begin{cases} v & \text{if } v \in D \\ \mathfrak{g}(\mathfrak{h}^{-1}(v)) & \text{if } v \notin D. \end{cases} \end{aligned}$$

Because h returns each element of X at most once, it inherits injectivity from $\mathfrak{g}$. As a result, $\mathfrak{h}$ is well-defined and in fact an element of $R_{C|D}$.

If $v \in (\mathfrak{h} \,.\, C)\backslash D$, then $\mathfrak{h}^{-1}(v) \in C$. Now supposing $\mathfrak{g}(\mathfrak{h}^{-1}(v)) \in D$ leads to the contradiction that

$$v = \mathfrak{h}(\mathfrak{h}^{-1}(v)) = h(\mathfrak{h}^{-1}(v)) = \mathfrak{g}(\mathfrak{h}^{-1}(v)) \in D.$$

Therefore, $\mathfrak{g}(\mathfrak{h}^{-1}(v)) \in D$, which implies that i inherits injectivity from $\mathfrak{g}\mathfrak{h}^{-1}$ so that $\mathfrak{i}$ is well-defined. Moreover, for all $c \in C$,

$$\mathfrak{i}(\mathfrak{h}(c)) = i(\mathfrak{h}(c)) = \begin{cases} \mathfrak{h}(c) & \text{if } \mathfrak{h}(c) \in D \\ \mathfrak{g}(\mathfrak{h}^{-1}(\mathfrak{h}(c))) & \text{if } \mathfrak{h}(c) \notin D \end{cases} = \mathfrak{g}(c)$$

because $\mathfrak{h}(c) = h(c) = \mathfrak{g}(c)$ if $\mathfrak{h}(c) \in D$. Finally, for each $d \in D$ we have $\mathfrak{i}(d) = i(d) = d$. This validates the definition of $R_{C|D}$.

Next, we show that most successor words are redundant for the representation of an observation table approximation.

Proposition 7.20. *For each $s \in S$ and every $a \in A$ there is a $b \in (A \dashv t^{\dagger}_{\mathcal{L}}(s))$ such that $\xi^{\dagger}_{\delta}(s)(a) \sim \xi^{\dagger}_{\delta}(s)(b)$.*

Proof. Determine $b \in (A \dashv t^{\dagger}_{\mathcal{L}}(s))$ satisfying $(b, t^{\dagger}_{\mathcal{L}}(s)) \sim (a, t^{\dagger}_{\mathcal{L}}(s))$. That is, there is an $\mathfrak{i} \in \mathfrak{G}$ such that $\mathfrak{i} \,.\, b = a$ and $\mathfrak{i} \,.\, t^{\dagger}_{\mathcal{L}}(s) = t^{\dagger}_{\mathcal{L}}(s)$. Then, for each $\hat{e} \in \widehat{E}$,

$$\begin{aligned} \xi^{\dagger}_{\delta}(s)(a)(\hat{e}) &= t^{\dagger}_{\mathcal{L}}(s)(a\hat{e}) \\ &= (\mathfrak{i} \,.\, t^{\dagger}_{\mathcal{L}}(s))(a\hat{e}) \\ &= \mathfrak{i} \,.\, t^{\dagger}_{\mathcal{L}}(s)(\mathfrak{i}^{-1} \,.\, a\hat{e}) \\ &= \mathfrak{i} \,.\, \xi^{\dagger}_{\delta}(s)(\mathfrak{i}^{-1} \,.\, a)(\mathfrak{i}^{-1} \,.\, \hat{e}) \\ &= \mathfrak{i} \,.\, \xi^{\dagger}_{\delta}(s)(b)(\mathfrak{i}^{-1} \,.\, \hat{e}) \\ &= (\mathfrak{i} \,.\, \xi^{\dagger}_{\delta}(s)(b))(\hat{e}). \end{aligned}$$

□

Corollary 7.21. *An observation table approximation $\widehat{\mathsf{T}}(S, E)$ is closed if and only if for all $s \in S$ and $a \in (A \dashv t^{\dagger}_{\mathcal{L}}(s))$ there is an $s' \in S$ such that $\xi^{\dagger}(s') \sim \xi^{\dagger}_{\delta}(s)(a)$.*

To ensure practical applicability of these results, we can of course replace $A \dashv t^{\dagger}_{\mathcal{L}}(s)$ by $A \dashv C$ for any finite $C \subseteq \mathcal{V}$ supporting $t^{\dagger}_{\mathcal{L}}(s)$. As we show next, a similar technique can be used to already simplify consistency, but it still involves iterating over an infinite number of group elements. We will come back to this later.

Proposition 7.22. *An observation table approximation* $\widehat{\mathsf{T}}(S, E)$ *is consistent if and only if for all* $s_1, s_2 \in S$ *and* $\mathfrak{g} \in \mathfrak{G}$ *such that* $\mathfrak{g} \,.\, \xi^{\dagger}(s_1) = \xi^{\dagger}(s_2)$ *we have for all* $a \in (A \dashv (t^{\dagger}_{\mathcal{L}}(s_1), t^{\dagger}_{\mathcal{L}}(s_2)))$ *that* $\mathfrak{g} \,.\, \xi^{\dagger}_{\delta}(s_1)(b) = \xi^{\dagger}_{\delta}(s_2)(b)$.

Proof. If the approximation is consistent, the above property certainly holds, since $(A \dashv (t^{\dagger}_{\mathcal{L}}(s_1), t^{\dagger}_{\mathcal{L}}(s_2))) \subseteq A$.

Conversely, assume that the above property holds and that certain $\mathfrak{g} \in \mathfrak{G}$ and $s_1, s_2 \in S$ satisfy $\xi(\mathfrak{g}, s_1) = \xi^{\dagger}(s_2)$. Given $a \in A$, determine $b \in (A \dashv (t^{\dagger}_{\mathcal{L}}(s_1), t^{\dagger}_{\mathcal{L}}(s_2))) \subseteq A$ and $\mathfrak{i} \in \mathfrak{G}$ such that $\mathfrak{i} \,.\, b = a$, $\mathfrak{i} \,.\, t^{\dagger}_{\mathcal{L}}(s_1) = t^{\dagger}_{\mathcal{L}}(s_1)$, and $\mathfrak{i} \,.\, t^{\dagger}_{\mathcal{L}}(s_2) = t^{\dagger}_{\mathcal{L}}(s_2)$. Repeating the calculation from the proof of Proposition 7.20, we obtain $\xi^{\dagger}_{\delta}(s_1)(a) = \mathfrak{i} \,.\, \xi^{\dagger}_{\delta}(s_1)(b)$ and $\xi^{\dagger}_{\delta}(s_2)(a) = \mathfrak{i} \,.\, \xi^{\dagger}_{\delta}(s_2)(b)$, but we also have $\mathfrak{i} \,.\, \xi^{\dagger}(s_1) = \xi^{\dagger}(s_1)$ and $\mathfrak{i} \,.\, \xi^{\dagger}(s_2) = \xi^{\dagger}(s_2)$. Letting $\mathfrak{i}^{-1}$ act on both sides, we get $\xi^{\dagger}(s_2) = \mathfrak{i}^{-1} \,.\, \xi^{\dagger}(s_2)$. Since

$$\mathfrak{g}\mathfrak{i} \,.\, \xi^{\dagger}(s_1) = \mathfrak{g} \,.\, \xi^{\dagger}(s_1) = \xi^{\dagger}(s_2) = \mathfrak{i}^{-1} \,.\, \xi^{\dagger}(s_2),$$

we have $\mathfrak{i}^{-1}\mathfrak{g}\mathfrak{i} \,.\, \xi^{\dagger}(s_1) = \xi^{\dagger}(s_1)$. From the assumption it then follows that $\mathfrak{i}^{-1}\mathfrak{g}\mathfrak{i} \,.\, \xi^{\dagger}_{\delta}(s_1)(b) = \xi^{\dagger}_{\delta}(s_2)(b)$, so we conclude that

$$\mathfrak{g} \,.\, \xi^{\dagger}_{\delta}(s_1)(a) = \mathfrak{g}\mathfrak{i} \,.\, \xi^{\dagger}_{\delta}(s_1)(b) = \mathfrak{i} \,.\, \xi^{\dagger}_{\delta}(s_2)(b) = \xi^{\dagger}_{\delta}(s_2)(a). \qquad \square$$

Regarding the columns, we show that for each row and every $e \in E$ only the outputs for a finite number of permutations of e need to be recorded to represent the infinite number of cells. That is, we can choose $\mathsf{rep}(\widehat{E}) \subseteq E$, and then the required permutations are given by the proposition. It is clear that $\widehat{E}$ is nominal—each word is supported by the union of supports for its symbols.

Proposition 7.23. *For each $f \in Y^{\widehat{E}}$ finitely supported by $C \subseteq \mathcal{V}$ and every $\hat{e} \in \widehat{E}$ there is an $\hat{e}' \in (\widehat{E} \dashv C)$ such that $f(\hat{e}') \sim f(\hat{e})$.*

Proof. Determine $\hat{e}' \in (\widehat{E} \dashv C)$ and $\mathfrak{i} \in \mathfrak{G}$ such that $\mathfrak{i} \,.\, \hat{e}' = \hat{e}$ and $\mathfrak{i} \,.\, f = f$. Then $f(\hat{e}) = f(\mathfrak{i} \,.\, \hat{e}') = \mathfrak{i} \,.\, f(\hat{e}')$. □

Important is that the above proof is constructive: given $\hat{e}$, we can actually find $\hat{e}' \in (\widehat{E} \dashv C)$ and $\mathfrak{i} \in \mathfrak{G}$ such that $\mathfrak{i} \,.\, f(\hat{e}') = f(\hat{e})$.

Finally, we need to be able to determine equality of rows.

Proposition 7.24. *If $f \in Y^{\widehat{E}}$ is finitely supported by $C \subseteq \mathcal{V}$ and $g \in Y^{\widehat{E}}$ is finitely supported by $D \subseteq \mathcal{V}$, then $f = g$ if and only if for each $\hat{e} \in (\widehat{E} \dashv C \cup D)$ we have $f(\hat{e}) = g(\hat{e})$.*

Proof. Assume that for each $\hat{e} \in (\widehat{E} \dashv C \cup D)$ we have $f(\hat{e}) = g(\hat{e})$. Given $\hat{e} \in \widehat{E}$, we can find $\hat{e}' \in (\widehat{E} \dashv C \cup D)$ and $\mathfrak{i} \in \mathfrak{G}$ satisfying $\mathfrak{i} \,.\, \hat{e}' = \hat{e}$, $\mathfrak{i} \,.\, f = f$, and $\mathfrak{i} \,.\, g = g$, noting that $C \cup D$ supports both f and g. Hence,

$$f(\hat{e}) = f(\mathfrak{i} \,.\, \hat{e}') = \mathfrak{i} \,.\, f(\hat{e}') = \mathfrak{i} \,.\, g(\hat{e}') = g(\mathfrak{i} \,.\, \hat{e}') = g(\hat{e}).$$

The converse is trivial. □

Using the results of this section, the observation structure corresponding to $\widehat{\mathsf{T}}(S, E)$ can be represented as follows. As in Section 6.1, the upper part of our table has a row for each $s \in S$. For the lower part, however, we use Proposition 7.20 to have a row only for each sa with $s \in S$ and $a \in (A \dashv t^{\dagger}_{\mathcal{L}}(s))$. Similarly, for a row $f \in Y^{\widehat{E}}$ belonging to a row in the table—which due to Proposition 7.17 is finitely supported—we do not give the output for each suffix in $\widehat{E}$, as there would in general be an infinite number of such suffixes; instead, we apply Proposition 7.23 so that only the suffixes in $\widehat{E} \dashv C$ have to be considered, where some finite $C \subseteq \mathcal{V}$ supports f. Note that the suffixes that are to be considered in this way depend on the specific row. Therefore, we label columns by the elements $e \in E$ and for each specific row f list multiple values in the cell corresponding to e. Specifically, such a cell will contain for each $\mathfrak{h} \in R_{e|C}$

an entry of the form $\mathfrak{h} \mathbin{.} e$: $f(\mathfrak{h} \mathbin{.} e)$. Note that this means that we choose $\mathsf{rep}(\widehat{E}) \subseteq E$. The entries will be comma-separated and may be spread over multiple lines.

For example, consider a countably infinite set $\mathcal{V} = \{a, b, c, \ldots\}$ and the alphabet $A = \mathcal{V}$, with an action given by application. Let the language $\mathcal{L}^\dagger$ be

$$\{xvvy \mid x, y \in A^*, v \in A\}.$$

Note that $C_{t^\dagger_{\mathcal{L}}(\varepsilon)} = \emptyset$: for each $\mathfrak{g} \in \mathfrak{G}$, $\mathfrak{g} \mathbin{.} \mathcal{L}^\dagger = \mathcal{L}^\dagger$. This also means that $\mathcal{L}\colon A^* \to 2$ is actually an equivariant function (which we do not require in general). In Table 7.4a is shown a representation of the observation structure for $\widehat{\mathsf{T}}(\{\varepsilon\}, \{\varepsilon, a\})$. This approximation is not closed: $\xi(\mathfrak{g}, \varepsilon)(a) = 0$ for all $\mathfrak{g} \in \mathfrak{G}$, but $\xi^\dagger_\delta(\varepsilon)(a)(a) = 1$. The table that results from adding a to S is still not closed: $\xi^\dagger_\delta(a)(a)(\varepsilon) = 1$, but $\xi^\dagger(\varepsilon)(\varepsilon) = \xi^\dagger(a)(\varepsilon) = 0$. Adding also aa to S yields Table 7.4b. This table *is* closed: we see that $\xi^\dagger_\delta(aa)(a) = \xi^\dagger_\delta(aa)(b) = \xi^\dagger(aa)$ and $\xi^\dagger_\delta(a)(b) = \xi((ab), a)$.

Let us expand on how this last fact could be determined mechanically. Specifically, we want to find out whether $\xi^\dagger(a) \sim \xi^\dagger_\delta(a)(b)$. The first step is to determine the least supports of $\xi^\dagger(a)$ and $\xi^\dagger_\delta(a)(b)$. We know by Proposition 7.17 that

$$\begin{aligned} C_{\xi^\dagger(a)} &\subseteq C_{t^\dagger_{\mathcal{L}}(\varepsilon)} \cup C_a \ = \{a\} \\ C_{\xi^\dagger_\delta(a)(b)} &\subseteq C_{t^\dagger_{\mathcal{L}}(\varepsilon)} \cup C_{ab} = \{a, b\}. \end{aligned}$$

Note that this knowledge was already used to determine the representation of these rows. We use Proposition 7.18 to prune the supports as much as possible, until we reach the least supports. Regarding $\xi^\dagger(a)$, we have $a \in C_{\xi^\dagger(a)}$ if and only if $(ab) \mathbin{.} \xi^\dagger(a) \neq \xi^\dagger(a)$. Recall from the generic facts about supports that $(ab) \mathbin{.} \xi^\dagger(a)$ is supported by $(ab) \mathbin{.} \{a\} = \{b\}$, so by Proposition 7.24 we have $(ab) \mathbin{.} \xi^\dagger(a) = \xi^\dagger(a)$ if and only if $\xi((ab), a)(\mathfrak{h} \mathbin{.} e) = \xi^\dagger(a)(\mathfrak{h} \mathbin{.} e)$ for

	ε	a
ε	ε: 0	a: 0
a	ε: 0	a: 1, b: 0

(a) Closedness defect

	ε	a
ε	ε: 0	a: 0
a	ε: 0	a: 1, b: 0
aa	ε: 1	a: 1, b: 1
ab	ε: 0	a: 0, b: 1, c: 0
aaa	ε: 1	a: 1, b: 1
aab	ε: 1	a: 1, b: 1, c: 1

(b) Closed table

Table 7.4: Tables for the language $\{xvvy \mid x, y \in A^*, v \in A\}$

each $\hat{e} \in (\widehat{E} \dashv \{a, b\})$. Since

$$R_{\varepsilon|\{a,b\}} = \{\mathfrak{e}\} \qquad\qquad R_{a|\{a,b\}} = \{\mathfrak{e}, (ab), (ac)\},$$

we have $\widehat{E} \dashv \{a, b\} = \{\varepsilon, a, b, c\}$ and calculate

$$\begin{aligned} \xi((ab), a)(\varepsilon) &= 0 = \xi^\dagger(a)(\varepsilon) \\ \xi((ab), a)(a) &= 0 \neq 1 = \xi^\dagger(a)(a) \\ \xi((ab), a)(b) &= 1 \neq 0 = \xi^\dagger(a)(b) \\ \xi((ab), a)(c) &= 0 = \xi^\dagger(a)(c) \end{aligned}$$

We list all the calculations, but on discovering the first inequality we could have concluded immediately that $C_{\xi^\dagger(a)} = \{a\}$. Using the same procedure on each element of the known support for $\xi^\dagger_\delta(a)(b)$, we find that $C_{\xi^\dagger_\delta(a)(b)} = \{b\}$. To apply Proposition 7.13, we simply observe that there is a unique bijection $C_{\xi^\dagger(a)} \to C_{\xi^\dagger_\delta(a)(b)}$, which by Proposition 7.14 extends to $(ab) \in \mathfrak{G}$. We apply once more Proposition 7.24 to find that indeed $(ab) \,.\, \xi^\dagger(a) = \xi^\dagger_\delta(a)(b)$.

Consider for the same alphabet and output object the language

$$\{x, y\} \cup \{ax \mid a \in A \setminus \{x, y\}\},$$

	ε	a
ε	ε: 0	a: 0, x: 1, y: 1
a	ε: 0	a: 0, x: 1, y: 0
$\vdots$	$\vdots$	$\vdots$

(a) Consistency defect

	ε	a	ax
ε	ε: 0	a: 0, x: 1, y: 1	ax: 1, ay: 0, ab: 0, xa: 0, xy: 0, ya: 0, yx: 0
$\vdots$	$\vdots$	$\vdots$	$\vdots$

(b) Inconsistency resolved

Table 7.5: Tables for the language $\{x, y\} \cup \{ax \mid a \in A \setminus \{x, y\}\}$

with $x, y \in A$ fixed and $x \neq y$. In this case, $C_{t_{\mathcal{L}}} = \{x, y\}$, which we assume known in advance. The approximation $\widehat{\mathsf{T}}(\{\varepsilon, a, x, aa\}, \{\varepsilon, a\})$ reaches the minimal automaton (in $\mathbf{EM}(\mathfrak{G} \times (-))$) for the language and is thus closed and initialized. It is also responsive and, interestingly, satisfies $C_{\xi^\dagger(s)} = C_{t^\dagger_{\mathcal{L}}(s)}$ for each $s \in S$. This means that the supports of the rows cannot increase any further. Still, there is a consistency defect, which can be seen in the partial representation in Table 7.5a. We have $\xi((xy), \varepsilon) = \xi(\mathfrak{e}, \varepsilon)$, but $\xi_\delta((xy), \varepsilon)(a)(x) = 0 \neq 1 = \xi_\delta(\mathfrak{e}, \varepsilon)(a)(x)$. Adding the suffix ax to E resolves the inconsistency, as can be seen from Table 7.5b: the equality $\xi((xy), \varepsilon) = \xi(\mathfrak{e}, \varepsilon)$ does not hold anymore.

Such consistency defects that do not lead to an increase in the number of orbits or the size of the support of a certain row can still be detected automatically. The following result is the key to this. It allows us to eliminate the last infinite ingredient from Proposition 7.22.

Proposition 7.25. *If $x_1, x_2 \in U$ and $y_1, y_2 \in V$ for $\mathfrak{G}$-sets U and V are finitely supported and $\mathfrak{g} \in \mathfrak{G}$ is such that $\mathfrak{g} \,.\, x_1 = x_2$ and*

$\mathfrak{g} \cdot y_1 \neq y_2$, then there is an $\mathfrak{h} \in R_{(x_1,y_1)|(x_2,y_2)}$ such that $\mathfrak{h} \cdot x_1 = x_2$ and $\mathfrak{h} \cdot y_1 \neq y_2$.

Proof. We can determine $\mathfrak{h} \in R_{(x_1,y_1)|(x_2,y_2)}$ and $\mathfrak{i} \in \mathfrak{G}$ such that $\mathfrak{i}\mathfrak{h} \cdot x_1 = \mathfrak{g} \cdot x_1$, $\mathfrak{i}\mathfrak{h} \cdot y_1 = \mathfrak{g} \cdot y_1$, $\mathfrak{i} \cdot x_2 = x_2$, and $\mathfrak{i} \cdot y_2 = y_2$. We have $\mathfrak{i}\mathfrak{h} \cdot x_1 = \mathfrak{g} \cdot x_1 = x_2$, so

$$\mathfrak{h}^{-1} \cdot x_2 = \mathfrak{h}^{-1}\mathfrak{i}^{-1} \cdot x_2 = x_1,$$

from which $\mathfrak{h} \cdot x_1 = x_2$ follows.

Similarly, we find $\mathfrak{i}\mathfrak{h} \cdot y_1 = \mathfrak{g} \cdot y_1 \neq y_2$, and therefore

$$\mathfrak{h}^{-1} \cdot y_2 = \mathfrak{h}^{-1}\mathfrak{i}^{-1} \cdot y_2 \neq y_1.$$

We conclude that $\mathfrak{h} \cdot y_1 \neq y_2$. □

Corollary 7.26. *An observation table approximation $\widehat{\mathsf{T}}(S, E)$ is responsive if and only if for all access strings $s_1, s_2 \in S$ and every $\mathfrak{h} \in R_{(\xi^\dagger(s_1),\mathcal{L}^\dagger(s_1))|(\xi^\dagger(s_2),\mathcal{L}^\dagger(s_2))}$ such that $\mathfrak{h} \cdot \xi^\dagger(s_1) = \xi^\dagger(s_2)$ we have $\mathfrak{h} \cdot \mathcal{L}^\dagger(s_1) = \mathcal{L}^\dagger(s_2)$.*

Corollary 7.27. *An observation table approximation $\widehat{\mathsf{T}}(S, E)$ is consistent if and only if for all $s_1, s_2 \in S$, $a \in (A \dashv (t^\dagger_{\mathcal{L}}(s_1), t^\dagger_{\mathcal{L}}(s_2)))$, and $\mathfrak{h} \in R_{(\xi^\dagger(s_1),\xi^\dagger_\delta(s_1)(a))|(\xi^\dagger(s_2),\xi^\dagger_\delta(s_2)(a))}$ with $\mathfrak{h} \cdot \xi^\dagger(s_1) = \xi^\dagger(s_2)$ we have $\mathfrak{h} \cdot \xi^\dagger_\delta(s_1)(a) = \xi^\dagger_\delta(s_2)(a)$.*

Now that we know how to extend a table such that the hypothesis can be constructed, we want to actually construct that hypothesis, or at least an automaton equivalent to it. A scoop automaton provided by Proposition 6.21 will have a finite number of states. Since $\mathfrak{G} \times S$ is now in general infinite, it may not be immediately obvious that there is a computable right inverse $H \to \mathfrak{G} \times S$ of the function $e \colon \mathfrak{G} \times S \to H$. However, note that for each $h \in H$ there exists an $s \in S$ such that $e^\dagger(s) \sim h$. Using the procedure outlined earlier, we can really determine for each $s \in S$ whether there is a $\mathfrak{g} \in \mathfrak{G}$ such that $\mathfrak{g} \cdot e^\dagger(s) = h$, and if it exists we can find such a $\mathfrak{g}$. The task is thus reduced to finding a right inverse of $e^\dagger \colon S \to \mathsf{im}(e^\dagger)$. This function has a finite domain.

Although the state space has become finite, there may still be an infinite number of transitions. We outline now with a variation on Proposition 6.19 that it is sufficient to represent only a finite number of them.

Proposition 7.28. *If Q is an automaton in $\mathbf{EM}(\mathfrak{G} \times (-))$, (X, i, d) is a scoop for Q, and $C \subseteq \mathcal{V}$ supports every $\delta_Q(i(x))$ for $x \in X$, then the automaton $\tilde{X} = \mathfrak{G} \times X$ implicated below accepts the same language as Q.*

$$\mathsf{init}^\dagger_{\tilde{X}} = d \circ \mathsf{init}^\dagger_Q \qquad\qquad \mathsf{out}^\dagger_{\tilde{X}} = \mathsf{out}_Q \circ i$$

$$\delta^\dagger_{\tilde{X}}(x)(a) = \mathfrak{j} \,.\, (d^A \circ \delta_Q \circ i)(x)(b),$$

where $\mathfrak{j} \in \mathfrak{G}$ and $b \in (A \dashv C)$ are chosen[10] *such that $\mathfrak{j} \,.\, b = a$ and $\mathfrak{j} \,.\, c = c$ for all $c \in C$.*

Proof. We claim just as in the proof of Proposition 6.19 that $i^\sharp$ is an automaton homomorphism $\tilde{X} \to Q$. Since the initial state and output maps are the same as in that proposition, we only need to show that $\delta_Q \circ i = (i^\sharp)^A \circ \delta^\dagger_{\tilde{X}}$. For $x \in X$ and $a \in A$, let $\mathfrak{j} \in \mathfrak{G}$ and $b \in (A \dashv C)$ be such that $\mathfrak{j} \,.\, b = a$ and $\mathfrak{j} \,.\, c = c$ for all $c \in C$. Because

[10] A choice really has to be fixed here, for otherwise $\delta^\dagger_{\tilde{X}}$ may not be well-defined, as d^A may not be equivariant.

C supports $\delta_Q(i(x))$, it follows that $\mathfrak{j} \,.\, \delta_Q(i(x)) = \delta_Q(i(x))$. We have

$$\begin{aligned}
\delta_Q(i(x))(a) &= (\mathfrak{j} \,.\, \delta_Q(i(x)))(a) \\
&= \mathfrak{j} \,.\, (\delta_Q \circ i)(x)(\mathfrak{j}^{-1} \,.\, a) && \text{(lifting of } (-)^A\text{)} \\
&= \mathfrak{j} \,.\, (\delta_Q \circ i)(x)(b) && (\mathfrak{j} \,.\, b = a) \\
&= \mathfrak{j} \,.\, ((i^\sharp)^A \circ d^A \circ \delta_Q \circ i)(x)(b) && \text{(scoop property)} \\
&= \mathfrak{j} \,.\, i^\sharp((d^A \circ \delta_Q \circ i)(x)(b)) \\
&= i^\sharp(\mathfrak{j} \,.\, (d^A \circ \delta_Q \circ i)(x)(b)) && \text{(equivariance of } i^\sharp\text{)} \\
&= i^\sharp(\delta^\dagger_{\tilde{X}}(x)(a)) && \text{(definition of } \delta^\dagger_{\tilde{X}}\text{)} \\
&= ((i^\sharp)^A \circ \delta^\dagger_{\tilde{X}})(x)(a). && \square
\end{aligned}$$

Such a modified scoop automaton allows us to restrict the transition function to $\delta^\dagger_{\tilde{X}} \colon X \to (\mathfrak{G} \times X)^{A \dashv C}$. If X is finite, this function can be represented by finite means. A further optimization could be achieved by switching from a single set C to a function $C \colon X \to \mathcal{P}_{\text{fin}}(\mathcal{V})$ such that $C(x)$ supports $\delta_Q(i(x))$ for each $x \in X$—this would allow us to restrict to a family of functions $\delta^\dagger_{\tilde{X}}(x) \colon (A \dashv C(x)) \to \mathfrak{G} \times X$ for $x \in X$.

Consider the hypothesis H and its scoop $(S, e^\dagger, j)$ for some right inverse j of e. We need a finite set $C \subseteq \mathcal{V}$ that supports $\delta_H(e^\dagger(s))$ for all $s \in S$. By the definition of δ_H, this says C supports $\mathsf{close}^\dagger(s)$ for all $s \in S$. Because by closedness $\xi^\dagger_\delta = m^A \circ \mathsf{close}^\dagger$ and m^A is just an inclusion, we need C to support $\xi^\dagger_\delta(s)$ for all $s \in S$. A simple upper bound follows from the result below. Least supports could then be found similar to how least supports were found for rows.

Proposition 7.29. *For each $s \in S$, $\xi^\dagger_\delta(s)$ is supported by $C_{t^\dagger_{\mathcal{L}}(\varepsilon)} \cup C_s$.*

Proof. Suppose $\mathfrak{g} \in \mathfrak{G}$ is such that $\mathfrak{g} \,.\, c = c$ for all $c \in C_{t^\dagger_{\mathcal{L}}(\varepsilon)} \cup C_s$.

Thus, $\mathfrak{g} \mathbin{.} t^{\dagger}_{\mathcal{L}}(\varepsilon) = t^{\dagger}_{\mathcal{L}}(\varepsilon)$ and $\mathfrak{g} \mathbin{.} s = s$.

$$\begin{aligned}
(\mathfrak{g} \mathbin{.} \xi^{\dagger}_{\delta}(s))(a)(\hat{e}) &= \mathfrak{g} \mathbin{.} t^{\dagger}_{\mathcal{L}}(s)(\mathfrak{g}^{-1} \mathbin{.} a\hat{e}) \\
&= \mathfrak{g} \mathbin{.} t^{\dagger}_{\mathcal{L}}(\varepsilon)(s \cdot (\mathfrak{g}^{-1} \mathbin{.} a\hat{e})) \\
&= \mathfrak{g} \mathbin{.} t^{\dagger}_{\mathcal{L}}(\varepsilon)(\mathfrak{g}^{-1} \mathbin{.} sa\hat{e}) \\
&= (\mathfrak{g} \mathbin{.} t^{\dagger}_{\mathcal{L}}(\varepsilon))(sa\hat{e}) \\
&= t^{\dagger}_{\mathcal{L}}(\varepsilon)(sa\hat{e}) \\
&= \xi^{\dagger}_{\delta}(s)(a)(\hat{e}). \qquad \square
\end{aligned}$$

7.3 Termination

We have already seen how the observation structure of $\widehat{T}(S, E)$ can be represented for finite sets of words S and E. We also know how to fix defects in the constructability of the hypothesis, and how to process a counterexample (since we can just apply the method explained in Section 6.2). It remains to be seen that the total number of such changes is bounded. At this point we assume that the minimal automaton M—the factorization of $t_{\mathcal{L}} \colon \mathfrak{G} \times A^* \to 2^{A^*}$—is orbit-finite.

In fixing a closedness or initialization defect, an orbit is added to the image of ξ. Therefore, the number of closedness and initialization fixes is bounded by the number of orbits of M. If a counterexample is processed, we either create an initialization or closedness defect, or we distinguish two permutations of rows that were previously equal. Specifically, in this case there are $\mathfrak{g} \in \mathfrak{G}$ and $s_1, s_2 \in S$ such that before adding the column we had $\mathfrak{g} \mathbin{.} \xi^{\dagger}(s_1) = \xi^{\dagger}(s_2)$, but afterwards we have $\mathfrak{g} \mathbin{.} \xi^{\dagger}(s_1) \neq \xi^{\dagger}(s_2)$. Note that this same situation occurs when fixing a responsiveness or consistency defect.

If after adding the new column there is no $\mathfrak{j} \in \mathfrak{G}$ at all such that $\mathfrak{j} \mathbin{.} \xi^{\dagger}(s_1) = \xi(s_2)$, then we must have found a new orbit—$\xi^{\dagger}(s_1)$ and $\xi^{\dagger}(s_2)$ used to be in the same orbit, but this relation has been broken. Suppose there does exist a $\mathfrak{j} \in \mathfrak{G}$ such that $\mathfrak{j} \mathbin{.} \xi^{\dagger}(s_1) = \xi(s_2)$. Note that this equation holds also in the old situation, as we have only added a column. Then in the old situation

$$\mathfrak{j}\,.\,\xi^\dagger(s_1) = \xi(s_2) = \mathfrak{g}\,.\,\xi^\dagger(s_1),$$

but in the new situation

$$\mathfrak{j}\,.\,\xi^\dagger(s_1) = \xi(s_2) \neq \mathfrak{g}\,.\,\xi^\dagger(s_1).$$

Let us simplify this a little. We now know that there are $s \in S$ and $\mathfrak{g} \in \mathfrak{G}$ such that before adding the column, $\mathfrak{g}\,.\,\xi^\dagger(s) = \xi^\dagger(s)$; but after adding the column, $\mathfrak{g}\,.\,\xi^\dagger(s) \neq \xi^\dagger(s)$. The following shows that such a $\mathfrak{g}$, if it exists, can be found in a fixed finite set.

Proposition 7.30. *For all $s \in A^*$ and $\mathfrak{g} \in \mathfrak{G}$, there is an $\mathfrak{h} \in R_{t^\dagger_{\mathcal{L}}(s) | t^\dagger_{\mathcal{L}}(s)}$ such that for all observation table approximations $\widehat{T}(S, E)$ with $s \in S$ we have*

$$\mathfrak{g}\,.\,\xi^\dagger(s) = \xi^\dagger(s) \iff \mathfrak{h}\,.\,\xi^\dagger(s) = \xi^\dagger(s).$$

Proof. Determine $\mathfrak{h} \in R_{t^\dagger_{\mathcal{L}}(s) | t^\dagger_{\mathcal{L}}(s)}$ and $\mathfrak{i} \in \mathfrak{G}$ such that for all $c \in C_{t^\dagger_{\mathcal{L}}(s)}$, $\mathfrak{i}\mathfrak{h}\,.\,c = \mathfrak{g}\,.\,c$ and $\mathfrak{i}\,.\,c = c$. Since $\xi^\dagger(s) = \pi_{\widehat{E}}(t^\dagger_{\mathcal{L}}(s))$, we know from Lemma 7.16 that $\xi^\dagger(s)$ is supported by $C_{t^\dagger_{\mathcal{L}}(s)}$. Thus, $\mathfrak{i}\mathfrak{h}\,.\,\xi^\dagger(s) = \mathfrak{g}\,.\,\xi^\dagger(s)$ and $\mathfrak{i}\,.\,\xi^\dagger(s) = \xi^\dagger(s)$. If $\mathfrak{g}\,.\,\xi^\dagger(s) = \xi^\dagger(s)$, then

$$\mathfrak{h}\,.\,\xi^\dagger(s) = \mathfrak{i}^{-1}\mathfrak{g}\,.\,\xi^\dagger(s) = \mathfrak{i}^{-1}\,.\,\xi^\dagger(s) = \xi^\dagger(s);$$

if $\mathfrak{g}\,.\,\xi^\dagger(s) \neq \xi^\dagger(s)$, then

$$\mathfrak{i}\mathfrak{h}\,.\,\xi^\dagger(s) = \mathfrak{g}\,.\,\xi^\dagger(s) \neq \xi^\dagger(s),$$

and therefore $\mathfrak{h}\,.\,\xi^\dagger(s) \neq \mathfrak{i}^{-1}\,.\,\xi^\dagger(s) = \xi^\dagger(s)$. □

We can now see that if $\mathfrak{g} \in \mathfrak{G}$ is such that in the old situation $\mathfrak{g}\,.\,\xi^\dagger(s) = \xi^\dagger(s)$, but in the new situation $\mathfrak{g}\,.\,\xi^\dagger(s) \neq \xi^\dagger(s)$, then there is an $\mathfrak{h} \in R_{t^\dagger_{\mathcal{L}}(s) | t^\dagger_{\mathcal{L}}(s)}$ such that in the old situation $\mathfrak{h}\,.\,\xi^\dagger(s) = \xi^\dagger(s)$ and in the new situation $\mathfrak{h}\,.\,\xi^\dagger(s) \neq \xi^\dagger(s)$. Since adding columns will not make unequal rows equal, this for each $s \in S$ happens at most $|R_{t^\dagger_{\mathcal{L}}(s) | t^\dagger_{\mathcal{L}}(s)}|$ times. We already know that S will not grow indefinitely, so the algorithm must terminate.

7.4 Discussion

The actual hypothesis in $\mathbf{EM}(\mathfrak{G} \times (-))$ is in fact a nominal automaton (the $\mathfrak{G}$-set of states is a nominal set), as the rows of the table are all finitely supported. Furthermore, the assumption that $t^\dagger_{\mathcal{L}}(\varepsilon)$ is finitely supported is equivalent to the minimal automaton for the language being nominal. Bojańczyk et al. [21] have developed an elaborate theory for the representation of nominal automata. Using their results, we could in the end represent the actual minimal automaton for the language that has been learned. Note that "minimal" does not necessarily imply that this representation is more succinct than an appropriate scoop automaton. In fact, representing the minimal automaton involves representing the algebra structure that we were able to avoid using scoops. One of the advantages of working with minimal automata is that they can more easily be compared: we know that two minimal automata accept the same language if and only if they are isomorphic. A practical description of the representation of the hypothesis as a nominal automaton is left as future work.

Learning algorithms for register automata [25] have been developed for several years now [40, 39, 23, 26]. It is to be expected that the languages accepted by such automata can also be accepted by certain orbit-finite nominal automata, as is the case for the related *finite memory automata* [21].

The register automaton learning algorithm by Howar et al. [40] in particular has many similarities with our observation table algorithm outlined in Section 7.2. One of the differences is that they do not always represent the entire function $\xi^\dagger_\delta$ in the table. For each newly added row in the upper part there is initially only one row in the lower part; the hypothesis is constructed with a partial transition function and additional transitions are learned through counterexamples. Their custom made canonical register automaton and specialized counterexample processing method also play a role here.

We have not provided a query complexity analysis for the algorithm discussed in this section for the simple reason that the obvious upper bounds are rather monstrous. That is, the sets $R_{C|D}$ for finite $C, D \subseteq \mathcal{V}$ are already in the equality symmetry very large, and filling the lower part of the table involves such sets in two dimensions: one for the successor words and one for the suffixes. We note that Howar et al. [40] also struggle with multiple exponential factors in their membership query complexity. It remains for future research to find out if this situation can be improved.

8 Conclusions

We have provided a unifying framework through which automata learning, minimization, and equivalence testing can by studied for various types of automata. Many results are derived on a completely abstract category theoretical level and thus potentially have applications that would otherwise not have been identified as instances of automata learning. On a concrete level, we have fitted into our framework the known automata learning concepts of observation tables and discrimination trees. Moreover, we have reviewed the lifting of many ingredients to categories of algebras for a monad, and we have given a few examples of learning algorithms in such settings. Most interestingly, this includes the category of $\mathfrak{G}$-sets for a group $\mathfrak{G}$. By allowing certain infinite groups, we have set the first steps towards a learning algorithm for nominal automata.

Other frameworks have been developed to study different automata learning algorithms [18, 42]. Especially the work of Isberner [42] provides a wealth of practical information. However, in both cases the results are not directly applicable to e.g. settings with additional structure such as explored in Section 6.

A preliminary investigation of generalizing concepts in automata learning using category theory was performed by Jacobs and Silva [46]. Although our abstract definitions of closedness and consistency are strongly based on theirs, they did not yet attempt to formulate anything like our approximations; instead, what we call the approximated response they define very concretely and just for the case of an observation table, leaving it unclear what should be allowed in an arbitrary setting.

Many directions for future research are left open. In this thesis we have avoided altogether the topic of hypothesis minimality. A side effect of using the counterexample processing method of Rivest and Schapire [61] is that the hypotheses generated by the algorithm may not be minimal [64]. In particular, this means that the approximation is not complete for its own hypothesis. As noted in Section 4.1, this property would be desired because it may lead to a seamless integration of conformance testing algorithms into the learning algorithm.

We remarked in Section 6.3 that the algorithm by Bollig et al. [22] is very similar to the algorithm we derive, except that their notion of consistency is weaker. A first step to understanding this may be a general investigation of relaxations on the properties used to construct a hypothesis. For instance, we have used several times a right inverse $i\colon H \to S$ of $e\colon S \to H$. Using such a map, one could bypass consistency and still construct a dynamics for the hypothesis as $\mathsf{close} \circ i$. Such a definition will however enjoy less desirable properties than the solid hypothesis that we have studied in this thesis (which we expect might help to explain why the algorithm by Bollig et al. would not terminate with the counterexample processing of Angluin). Similarly, one could cheat on closedness given a left inverse of $\mathcal{D}m\colon \mathcal{D}H \to \mathcal{D}P$.

Despite the negative results for passive learning mentioned in the introduction of Section 3, passive learning algorithms do exist [20, 56]; they simply do not produce optimal results. Future research could attempt to place these in our framework as well.

Although our abstract characterizations cover observation tables and discrimination trees in the classical case, we do not have abstract definitions of these. As a result, such concepts have to be redefined in new settings, as we did in $\mathbf{EM}(\mathfrak{G} \times (-))$, or at least lifted, as we did in $\mathbf{EM}(\mathcal{P})$. In both of these cases we considered only observation tables. Note that observation tables classify into 2^E, which is likely to have a structure similar to that of 2^{A^*}. Therefore, generalizations of observation tables can be expected to be more straightforward than generalizations of discrimination trees. It would be interesting to see if anything at all more efficient than observation tables could be used in these settings. As we have shown in Section 4, such results would be applicable also to minimization and conformance testing.

References

[1] Fides Aarts, Bengt Jonsson, and Johan Uijen. Generating models of infinite-state communication protocols using regular inference with abstraction. In *Testing Software and Systems*, volume 6435 of *LNCS*, pages 188–204. Springer, 2010.

[2] Fides Aarts, Faranak Heidarian, Harco Kuppens, Petur Olsen, and Frits Vaandrager. Automata learning through counterexample guided abstraction refinement. In *FM 2012: Formal Methods*, volume 7436 of *LNCS*, pages 10–27. Springer, 2012.

[3] Fides Aarts, Paul Fiterău-Broştean, Harco Kuppens, and Frits Vaandrager. Learning register automata with fresh value generation. In *Theoretical Aspects of Computing-ICTAC 2015*, volume 9399 of *LNCS*, pages 165–183. Springer, 2015.

[4] Jiří Adámek. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae*, 15(4):589–602, 1974.

[5] Jiří Adámek, Horst Herrlich, and George E. Strecker. Abstract and concrete categories: The joy of cats. 2004.

[6] Dana Angluin. A note on the number of queries needed to identify regular languages. *Information and control*, 51(1):76–87, 1981.

[7] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[8] Dana Angluin and Dana Fisman. Learning regular omega languages. In *Algorithmic Learning Theory*, volume 8776 of *LNCS*, pages 125–139. Springer, 2014.

[9] Dana Angluin, Sarah Eisenstat, and Dana Fisman. Learning regular languages via alternating automata. In *Proceedings*

of the 24th International Conference on Artificial Intelligence, pages 3308–3314. AAAI Press, 2015.

[10] Michael A. Arbib and Ernest G. Manes. Machines in a category: An expository introduction. *SIAM review*, 16(2):163–192, 1974.

[11] Michael A. Arbib and Ernest G. Manes. Foundations of system theory: decomposable systems. *Automatica*, 10(3):285–302, 1974.

[12] Michael A. Arbib and Ernest G. Manes. Adjoint machines, state-behavior machines, and duality. *Journal of Pure and Applied Algebra*, 6(3):313–344, 1975.

[13] Michael A. Arbib and Ernest G. Manes. Fuzzy machines in a category. *Bulletin of the Australian Mathematical Society*, 13 (02):169–210, 1975.

[14] Michael A. Arbib and H. Paul Zeiger. On the relevance of abstract algebra to control theory. *Automatica*, 5(5):589–606, 1969.

[15] Steve Awodey. *Category theory.* Oxford University Press, 2010.

[16] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*, volume 26202649. MIT press Cambridge, 2008.

[17] Adriana Balan and Alexander Kurz. On coalgebras over algebras. *Electronic Notes in Theoretical Computer Science*, 264 (2):47–62, 2010.

[18] 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*, pages 53–72. Springer, 1997.

[19] Therese Berg, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt, and Bernhard Steffen. On the correspondence

between conformance testing and regular inference. In *Fundamental Approaches to Software Engineering*, volume 3442 of *LNCS*, pages 175–189. Springer, 2005.

[20] Alan W. Biermann and Jerome A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE transactions on Computers*, (6):592–597, 1972.

[21] Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3):4, 2014.

[22] Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In *IJCAI*, volume 9, pages 1004–1009, 2009.

[23] Benedikt Bollig, Peter Habermehl, Martin Leucker, and Benjamin Monmege. A fresh approach to learning register automata. In *Developments in Language Theory*, volume 7907 of *LNCS*, pages 118–130. Springer, 2013.

[24] Filippo Bonchi, Marcello M. Bonsangue, Helle H. Hansen, Prakash Panangaden, Jan J.M.M. Rutten, and Alexandra Silva. Algebra-coalgebra duality in Brzozowski's minimization algorithm. *ACM Transactions on Computational Logic (TOCL)*, 15(1):3, 2014.

[25] Sofia Cassel, Falk Howar, Bengt Jonsson, Maik Merten, and Bernhard Steffen. A succinct canonical register automaton model. *Automated Technology for Verification and Analysis*, 6996:366–380, 2011.

[26] Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. Active learning for extended finite state machines. *Formal Aspects of Computing*, 28(2):233–263, 2016.

[27] Georg Chalupar, Stefan Peherstorfer, Erik Poll, and Joeri de Ruiter. Automated reverse engineering using Lego®. In

8th USENIX Workshop on Offensive Technologies (WOOT 14), 2014.

[28] Martin Chapman, Hana Chockler, Pascal Kesseli, Daniel Kroening, Ofer Strichman, and Michael Tautschnig. Learning the language of error. In *Automated Technology for Verification and Analysis*, volume 9364 of *LNCS*, pages 114–130. Springer, 2015.

[29] 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 *Proceedings of the 17th ACM conference on Computer and communications security*, pages 426–439. ACM, 2010.

[30] Tsun S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng.*, 4(3):178–187, 1978.

[31] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking.* MIT press, 1999.

[32] Joeri de Ruiter and Erik Poll. Protocol state fuzzing of TLS implementations. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 193–206, 2015.

[33] Hartmut Ehrig, Klaus-Dieter Kiermeier, Hans-Jörg Kreowski, and Wolfgang Kühnel. *Universal theory of automata: a categorical approach.* Vieweg+Teubner, 2013.

[34] Khaled El-Fakih, Roland Groz, Muhammad Naeem Irfan, and Muzammil Shahbaz. Learning finite state models of observable nondeterministic systems in a testing context. In *ICTSS*, pages 97–102, 2010.

[35] Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13(3-5):341–363, 2002.

[36] E. Mark Gold. System identification via state characterization. *Automatica*, 8(5):621–636, 1972.

[37] Bin-Lun Ho. *On effective construction of realizations from input-output descriptions.* PhD thesis, Stanford University, 1966.

[38] Falk Howar, Bernhard Steffen, and Maik Merten. Automata learning with automated alphabet abstraction refinement. In *Verification, Model Checking, and Abstract Interpretation*, volume 6538 of *LNCS*, pages 263–277. Springer, 2011.

[39] Falk Howar, Malte Isberner, Bernhard Steffen, Oliver Bauer, and Bengt Jonsson. Inferring semantic interfaces of data structures. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, volume 7609 of *LNCS*, pages 554–571. Springer, 2012.

[40] Falk Howar, Bernhard Steffen, Bengt Jonsson, and Sofia Cassel. Inferring canonical register automata. In *Verification, Model Checking, and Abstract Interpretation*, volume 7148 of *LNCS*, pages 251–266. Springer, 2012.

[41] Hardi Hungar, Oliver Niese, and Bernhard Steffen. Domain-specific optimization in automata learning. In *Computer Aided Verification*, volume 2725 of *LNCS*, pages 315–327. Springer, 2003.

[42] Malte Isberner. *Foundations of active automata learning: an algorithmic perspective.* PhD thesis, Technical University of Dortmund, 2015.

[43] Malte Isberner and Bernhard Steffen. An abstract framework for counterexample analysis in active automata learning. In *ICGI*, pages 79–93, 2014.

[44] Malte Isberner, Falk Howar, and Bernhard Steffen. The TTT algorithm: A redundancy-free approach to active automata learning. In *Runtime Verification*, volume 8734 of *LNCS*, pages 307–322. Springer, 2014.

[45] Malte Isberner, Falk Howar, and Bernhard Steffen. The open-source LearnLib. In *Computer Aided Verification*, volume 9206 of *LNCS*, pages 487–495. Springer, 2015.

[46] Bart Jacobs and Alexandra Silva. Automata learning: A categorical perspective. In *Horizons of the Mind. A Tribute to Prakash Panangaden*, volume 8464 of *LNCS*, pages 384–406. Springer, 2014.

[47] Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT press, 1994.

[48] Ali Khalili and Armando Tacchella. Learning nondeterministic Mealy machines. In *ICGI*, pages 109–123, 2014.

[49] David Lee and Mihalis Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994.

[50] David Lee and Mihalis Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[51] Saunders Mac Lane. *Categories for the working mathematician*, volume 5. Springer Science & Business Media, 1978.

[52] George H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[53] Edward F. Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.

[54] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[55] Oliver Niese. *An Integrated Approach to Testing Complex Systems*. PhD thesis, University of Dortmund, 2003.

[56] José Oncina and Pedro García. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition*, volume 5 of *Machine Perception and Artificial Intelligence*, pages 99–108. World Scientific, 1992.

[57] Warawoot Pacharoen, Toshiaki Aoki, Pattarasinee Bhattarakosol, and Athasit Surarerks. Active learning of nondeterministic finite state machines. *Mathematical Problems in Engineering*, 2013.

[58] Doron Peled, Moshe Y. Vardi, and Mihalis Yannakakis. Black box checking. *Journal of Automata, Languages and Combinatorics*, 7(2):225–246, 2002.

[59] Leonard Pitt and Manfred K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of the ACM (JACM)*, 40(1):95–142, 1993.

[60] Andrew M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57. Cambridge University Press, 2013.

[61] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.

[62] Jan J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[63] Mathijs Schuts, Jozef Hooman, and Frits Vaandrager. Refactoring of legacy software using model learning and equivalence checking: an industrial experience report. In *Integrated Formal Methods*, volume 9681 of *LNCS*, pages 311–325. Springer, 2016.

[64] Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to active automata learning from a practical perspective. In

Formal Methods for Eternal Networked Software Systems, volume 6659 of *LNCS*, pages 256–296. Springer, 2011.

[65] Gerrit Jan Tretmans. *A formal approach to conformance testing*. PhD thesis, University of Twente, 1992.

[66] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[67] M.P. Vasilevskii. Failure diagnosis of automata. *Cybernetics and Systems Analysis*, 9(4):653–665, 1973.

[68] Michele Volpato and Jan Tretmans. Active learning of nondeterministic systems from an ioco perspective. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, volume 8802 of *LNCS*, pages 220–235. Springer, 2014.