Normalization and Learning of Transducers on Trees and Words

Adrien Boiret

Ph.D. Thesis

to obtain the title of Ph.D. of Science
of the Doctoral School Sciences Pour l'Ingénieur
Université de Lille

Specialty : Computer Sciences

Inria Lille and Université de Lille (CRIStAL lab)
Links Team

defended on Nov 7, 2016

Jury :

Reviewers :	Helmut Seidl	-	TU Munich
	Olivier Carton	-	Université Paris Diderot
Advisor :	Joachim Niehren	-	Inria Lille
Co-advisor:	Aurélien Lemay	-	Université de Lille
Examiners :	Sebastian Maneth	-	University of Edinburgh
	Sophie Tison	-	Université de Lille

Abstract:

Since the arrival of the Web, various kinds of semi-structured data formats were introduced in the areas of computer science and technology relevant for the Web, such as document processing, database management, knowledge representation, and information exchange.

The most recent technologies for processing semi-structured data rely on the formats JSON and RDF. The main questions there are how to make semi-structured data dynamic and how to represent knowledge in data graphs, so that it can be extracted more easily. In this thesis, we study the conversion of semi-structured data from one schema to another. The most powerful solutions to this problem were proposed by the XML technology in the context of document processing. In the XML format, semi-structured data is restricted to data trees, so that schemas can be defined by tree automata (for instance in RELAXNG), possibly enhanced by constraints on data values (for instance by logical XPATH queries in SCHEMATRON). Document transformations can be defined in XSLT, a purely functional programming language with logical XPATH queries. The core of XSLT are macro tree transducers with navigation by XPATH queries.

We contribute new learning algorithms on tree transducers, that are based on methods from grammatical inference. Previous approaches were limited in their support of schema restrictions, lookaheads, and concatenation in the output. Aspects of these three limitations are addressed by our three main results:

1. We show how to learn deterministic top-down tree transducers with regular domain inspection ($\text{DTOPI}_{\text{reg}}$) in a Gold style learning model with limited resources. Our algorithm is based on a new normal forms for transducers in the class $\text{DTOPI}_{\text{reg}}$.

2. We show how to learn rational functions, described by deterministic transducers of words with lookahead (DWT_{ℓ}). We propose a novel normal form for such transducers which provides a compromise between lookahead and state minimization, and which enables a learning algorithm for the class DWT_{ℓ} in Gold's learning model with polynomial resources.

3. For the class of linear tree-to-word transducers with concatenation in the output, we present a normal form and show how to decide equivalence in polynomial time.

Résumé:

Le développement du Web a motivé l'apparition de nombreux types de formats de données semi-structurées pour les problèmes liés aux technologies du Web, comme le traitement des documents, la gestion de base de données, la représentation des connaissances, et l'échange d'informations.

Les technologies les plus récentes pour le traitement des données semi-structurées se basent sur les formats JSON et RDF. Les grandes questions y sont de rendre les données dynamiques, ou de représenter les connaissances sur les données semi-structurées afin de faciliter leur extraction. Dans cette thèse, nous étudions la conversion des données semi-structurées d'un schéma à un autre. Dans le cadre du traitement de documents, c'est la technologie XML qui offre la solution la plus puissante à ce problème. Dans le format XML, les données semi-structurée sont des arbres de données, dont les schémas peuvent être définis par des automates d'arbres (par exemple dans RELAXNG), possiblement renforcés par des contraintes sur les valeurs de données (par exemple par des requêtes logiques XPATH en SCHEMATRON). Les transformations de documents peuvent être spécifiées en XSLT, un langage de programmation purement fonctionnel muni de requêtes logiques XPATH. Le cœur de XSLT correspond aux transducteurs d'arbres à macros munis de la navigation par requêtes XPATH.

Nous proposons de nouveaux algorithmes pour l'apprentissage des transducteurs d'arbres, basés sur des méthodes de l'inférence grammaticale. Les approches précédentes se prêtent peu aux classes de transducteurs munis de restrictions de schéma, d'anticipations (lookahead), ou de concaténation dans la sortie. Nos trois résultats principaux abordent certains aspects de ces trois limitations:

1. Pour les transducteurs d'arbres de haut en bas déterministes avec une inspection de domaine régulière ($\text{DTOPI}_{\text{reg}}$), nous donnons un algorithme d'aprentissage dans le modèle de Gold avec des ressources limitées. Cet algorithme est basé sur une nouvelle forme normale pour la classe $\text{DTOPI}_{\text{reg}}$.

2. Nous montrons comment apprendre des fonctions rationnelles, décrites par les transducteurs de mots déterministes avec anticipation (DWT_ℓ). Nous proposons une nouvelle forme normale pour ces transducteurs qui fournit un compromis entre la minimisation de l'anticipation et du transducteur, et qui permet l'apprentissage de la classe DWT_ℓ dans le modèle de Gold avec des ressources polynomiales.

3. Pour la classe des transducteurs arbre-vers-mot linéaires, qui permet la concaténation dans sa sortie, nous présentons une forme normale, et montrons comment décider l'équivalence en temps polynomial.

Acknowledgments

For the rest of my life, I'll be refering to this document as "my PhD thesis". However, calling it mine shall not blind you to the fact that for this thesis to come to a conclusion, it took more than my efforts alone. I had help along the way, in many forms, from many people. I have here the opportunity to give them the credit and thanks I owe them.

I would first like to thank the reviewers of this thesis, Olivier Carton and Helmut Seidl. They provided hard work and patience in copious amounts to help me through the final stages of this document's production. I am also grateful to Sebastian Maneth and Sophie Tison for their role as examiners.

It goes without saying that I owe a lot of thanks to my PhD advisor, Joachim Niehren, and my co-advisor, Aurélien Lemay. I cannot imagine how hard it must be to both trust a PhD student enough to let him learn and grow, and offer guidance and advice to avoid errors and pitfalls all the while. In this regard, Joachim and Aurélien have my deepest gratitude for giving me enough trust and priceless opportunities to tackle interesting problems, to begin ambitious works, to grow as a researcher, while displaying treasures of patience to help me overcome the difficulties I encountered along the way.

This acknowledgment section would not be complete if it did not mention the members of Team Mostrare and Links I had the pleasure to work with. Their number does not allow me to name them all, but they all deserve my gratitude. "Team" is for them more than an administrative status. The Team was for me a friendly and stimulating workplace, where colleagues are true allies. The help and support this provides cannot be overstated.

During my thesis, I was given the opportunity to teach on the campus. This experience was made both delightful and enlightening by my colleagues at Télécom Lille and IUT Lille A. Their help was precious, and they have my sincerest thanks.

I would also like, as cliché as it may be, to thank my friends and my family. Their interest, their care, their support, was more help to make this thesis what it is than they may realize. I may mention especially my parents, but also Vincent and Grégoire, that had the misfortune to share my office and that I today consider friends. They kept me as sane as I get during the hardest of times, inside and outside of my research, and it means a lot to me.

To all of them my thanks, beyond what these few lines can express.

Contents

1 Introduction **1**

1.1 Motivation 1

1.1.1 Semi-Structured Data 1

1.1.2 XML Document Processing 2

1.1.3 Transformation Learning 6

1.1.4 Tree Transducers 8

1.2 Limitations of Transducer Learning 12

1.2.1 Schema Restrictions 12

1.2.2 Lookahead 13

1.2.3 Output Concatenation 14

1.3 Contributions 14

1.3.1 Learning DTOPs with Regular Inspection 15

1.3.2 Learning Rational Functions 17

1.3.3 Linear Tree to Word Transducers 18

2 Related Work on Transducers **21**

2.1 MSO Transformations 21

2.1.1 Words 22

2.1.2 Trees 23

2.2 Bottom-Up Transducers 24

2.3 Transducers for Data Trees 25

3 Preliminaries **27**

3.1 Words 27

3.1.1 Word Automata 28

3.1.2 Determinism 29

3.1.3 Myhill-Nerode Theorem 30

3.2 Trees 31

3.2.1 Tree Automata 32

3.2.2 Determinism 34

3.2.3 Myhill-Nerode Theorem 35

3.3 Automata Learning 36

3.3.1 Learning Model 37

3.3.2 Characteristic Samples 38

3.3.3 Learning Algorithm RPNI 40

4 A Learning Algorithm for Top-Down Tree Transducers 45
4.1 Introduction . . . 48
4.2 Illustration of Ideas . . . 52
4.2.1 Myhill-Nerode Theorem for DTops . . . 52
4.2.2 Example Transformation . . . 53
4.3 Top-Down Tree Transducers . . . 55
4.3.1 Top-Down Tree Transducers . . . 55
4.3.2 Top-Down Domains . . . 57
4.3.3 Domain Inspection . . . 58
4.4 Syntactic Equivalence . . . 60
4.4.1 Syntactic Alignment . . . 60
4.4.2 Trimmed Transducers . . . 62
4.4.3 Origins of Output Constructors . . . 63
4.4.4 Syntactic Equivalence . . . 64
4.5 Compatible Transducers . . . 65
4.6 Earliest Transducers . . . 68
4.7 Semantic Equivalence . . . 73
4.7.1 Semantic Alignments . . . 73
4.7.2 Semantic Equivalence . . . 75
4.7.3 Semantic Successors . . . 77
4.8 Unique Normal Forms . . . 80
4.9 Learning from Examples . . . 84
4.9.1 Learning Model . . . 84
4.9.2 Characteristic Samples . . . 85
4.9.3 Learning Algorithm . . . 90
4.10 Remaining Proofs . . . 95
4.10.1 Trimming a DTopI . . . 95
4.10.2 Origins of Output Constructors . . . 96
4.11 Conclusions . . . 99

5 Learning Top-Down Tree Transducers with Regular Inspection 101
5.1 Introduction . . . 102
5.2 Running Example . . . 102
5.3 Semantic Equivalence . . . 104
5.4 Unique Normal Forms . . . 107
5.5 Learning from Examples . . . 113
5.5.1 Characteristic Samples . . . 113
5.5.2 Learning Algorithm . . . 118

6 Learning Rational Functions 125
6.1 Introduction . . . 126
6.2 Rational Functions . . . 126
6.3 Transducers with Look-Ahead . . . 128
6.4 Building the Look-Ahead Automaton . . . 131
6.5 The Learning Algorithm . . . 133
6.6 Characteristic Samples . . . 134

7 Normal Forms of Linear Tree-to-Word Transducers 137
7.1 Introduction . . . 138
7.2 Linear Tree-to-Word Transducers . . . 139
7.3 Earliest Linear Transducers . . . 141
7.4 Reordering in Earliest Transducers . . . 143
7.4.1 Reordering Erasing States . . . 144
7.4.2 Reordering Producing States . . . 145
7.5 Myhill-Nerode Theorem and Normal Form . . . 148
7.6 Conclusion and Future Work . . . 149

8 Equivalence of Linear Tree-to-Word Transducers is in Polynomial Time 151
8.1 Introduction . . . 152
8.2 Preliminaries . . . 152
8.3 Linear Tree-to-Word Transducers . . . 153
8.4 Partial Normal Form . . . 155
8.4.1 Eliminating Non-Earliest Quasi-Periodic States . . . 156
8.4.2 Switching Periodic States . . . 163
8.4.3 Testing Equivalence in Polynomial Time . . . 164

9 Conclusion 167

Bibliography 169

Chapter 1
Introduction

1.1 Motivation

The way in which people and companies manage their data changed radically with the evolution of the Web. In the same time, the amount of data stored on machines is growing continuously, while the ways in which the data is used are diversifying. While relational databases continue to be largely used in most of the traditional web services, various alternative systems to manage semi-structured data [Abiteboul et al., 2011] were introduced, ranging from knowledge bases of Rdf triples [Allemang and Hendler, 2008, Powers, 2003], graph databases [Robinson et al., 2013], NoSql databases [Benzaken et al., 2013], and Xml databases [Powell, 2006].

1.1.1 Semi-Structured Data

The purpose of semi-structured data [Abiteboul, 1997] is to markup unstructured data such as numbers, texts, pictures, or videos with meta information, in order to add some structure that can then be used to search for data by database-like queries.

Data graphs are a natural model for semi-structured data that is used by graph databases [Robinson et al., 2013]. Data graphs are directed graphs with labels from a finite alphabet on nodes and edges, called the meta data. Besides these labels, the nodes may carry data values such a strings or numbers from an infinite repository. The concept of data graphs can also be seen as an instance of Rdf triple stores, as for knowledge graphs in the semantic Web [Allemang and Hendler, 2008].

Data trees are a little less powerful model for semi-structured data, since they restrict data graphs to be acyclic and connected. Data trees have a hierarchical structure that naturally arises in document processing, for instance in DocBook [Walsh, 2010] source files of technical documentations, in TEX [Knuth, 1986] source files of research papers, and in Html 5 [Pilgrim, 2010] for sources of websites. An example of a data tree representing a bibliography is given in Figure 1.1.

The *eXtendable Markup Language* (Xml) [Bray et al., 2008], and the *JavaScript Object Notation* (Json) [Bassett, 2015] are the two most prominent formats for data trees. On the Web, both formats are competing. While Xml

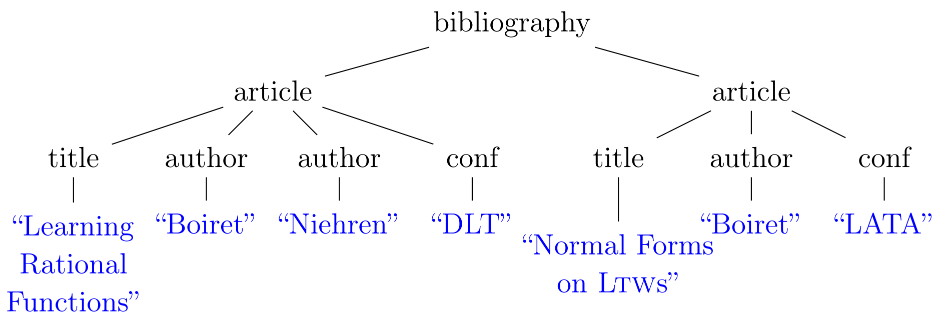


Figure 1.1: A data tree of an XML document representing a bibliography.

is used within HTML5 to model the static part, JSON is predominant when making Web pages dynamic by using JAVASCRIPT programs within HTML5. In the context of databases, JSON is used in NOSQL databases such as JAQL [Beyer et al., 2011], while XML was used in NOSQL databases such as EXIST [Siegel and Retter, 2014]. The precise format of data trees is irrelevant for the purpose of this thesis. However, the application domains of the XML technology are much closer to the problems we will study.

1.1.2 XML Document Processing

The biggest success of the XML technology is its usage for document processing, including documents on the Web or elsewhere. Besides this, XML is omnipresent as file exchange format between various programming languages and systems.

Document processing is usually done on a higher level of abstraction, by tools called content management systems [Barker, 2016] (Wikimedia, Plone, Wordpress, etc), or by document generators such as DOCBOOK. A prime idea of document processing is to separate the document's contents from its layout. In this way, the same document can be given multiple layouts. This requires to convert a data tree with the schema of the content into another data tree with the schema of the content with layout, such as the schema of HTML5. For example, one might want to convert the data tree for a bibliography in Figure 1.1 into the HTML5 data tree in Figure 1.2.

XML Schema

The first thing one needs for developing an document processing tool is a schema that specifies whether a document is valid for the considered application. For defining such schemas the XML technology provides the schema definition languages XML SCHEMA [van der Vlist, 2002], RELAXNG [van der Vlist, 2003], and SCHEMATRON [van der Vlist, 2007]. For instance,

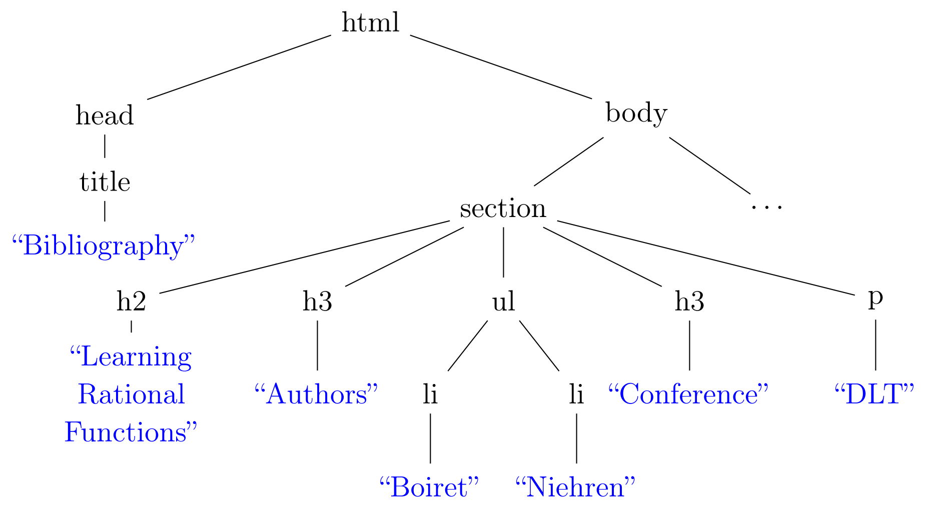


Figure 1.2: The data tree of an HTML5 document representing a layouted bibliography.

the schema for DocBook has been written in XML Schema and also in RelaxNG. Various communities have defined their own semi-structured data formats in these languages, including XBRL for business data, SPL for pharmaceutical products, or SBML for reaction networks in systems biology. A notable exception is HTML5 whose schema is defined differently even though following the XML data model. It should also be noticed that schema languages for JSON are only in a very early state [Pezoa et al., 2016].

The expressiveness of these three languages differ considerably. When ignoring constraints on data values and up to binary encodings, XML Schema supports regular tree languages definable by top-down deterministic tree automata [Martens et al., 2006], Schematron relies on regular tree languages definable in XPath [Benedikt and Koch, 2007] and thus first-order logic [Marx, 2005], while RelaxNG features all regular tree languages without restrictions [Hosoya, 2010].

XML Transformations

The second thing one needs for document processing is a way to convert data trees from one schema into another. This can be done with XSLT [Kay, 2001], which stands for XSL *Transformations* where XSL means XML *Stylesheet Language*.

The are many alternative solutions. The XML standards provide the XML *Query Language* XQuery as a second full-fledged tool of industrial strength. The use cases of XQuery are different from those of XSLT, in that XQuery targets database-like transformations as in SQL rather than document processing. But still, the overlap of both languages is remarkable. They share

XPath as a common core, but with slightly different semantics – ordered versus unordered node sets [Sebastian and Niehren, 2016] – and are often implemented on a common platform as with Saxon. Furthermore, it is possible to base Xslt and XQuery on a common core language such as X-Fun [Labath and Niehren, 2015].

There exists numerous alternatives to Xslt that were not standardized by the W3C. Most notably, the functional programming language CDuce [Benzaken et al., 2003], whose latest version does also extend on XPath [Castagna et al., 2015]. Another language worth mentioning is the early solution Fxt which is based on tree transducers [Neumann, 2000]. It should be noticed that no comparable transformation languages exist in the Json world. However, one can always rely on general purpose programming languages (Java, Caml, etc) for solving such transformation tasks.

Given that Xslt is strongly motivated by tree transducers, which are the main topic of the present PhD thesis, we illustrate the ideas of Xslt in some more detail. For this purpose, we present in Figure 1.3 a typical Xslt program, that converts the Xml document for the bibliography in Figure 1.1 into the Html document in Figure 1.2. In other words, we present an simple example for how to publish a document on the Web. The same program can be presented in transducer style [Maneth and Neven, 1999] by the pseudo-code in Figure 1.4. Given that this code is less verbose, we will explain this code rather than the Xslt program.

The Xslt program starts transforming the input tree from the root, and then produces its output tree in a top-down manner. The Xslt program has a single transformation mode. This mains that there are no auxilary transformation functions beside the main function, and hence the corresponding transducer has only a single state for the main function, that we call q here.

The transducer has 5 rules, one for each of the finitely many labels of the input tree. Each rule has an head, as for instance $q\langle bibliography\rangle$ that states what happens if state q is applied to a node of the input tree whose label matches the pattern *bibliography*. In this case, a sequence of trees and data values is produced in the output. This sequence starts with the tree $head(title(''Bibliography''))$ and is concatenated with the sequence obtained by applying the transducer to all children of the current node from the left to the right. The rule for $q\langle title\rangle$ is similar except that the output tree constructed by $h2(value{-}of(.))$, where $value{-}of(.)$ produces the concatenation of the data values of the input subtree rooted at the current *title*-node.

The transducer is called navigational in the sense that it can move from the current node to other nodes. In the present example the transducer will move to all the children, as in $q\langle child :: *\rangle$ from the left to the right, or to all children matching some label, as in $q\langle child :: title\rangle$. More generally an Xslt program may also navigate backwards in the input trees, by using an XPath

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="bibliography">
  <html>
    <head> <title>Bibliography</title> </head>
    <body> <xsl:apply-templates select=''child::*''/></
        body>
  </html>
</xsl:template>

<xsl:template match="article">
  <section>
    <xsl:apply-templates select="child::title"/>
    <h3> Authors </h3>
    <ul> <xsl:apply-templates select="child::author"/></ul
        >
    <xsl:apply-templates select="child::conf"/>
  </section>
</xsl:template>

<xsl:template match="title">
  <h2> <xsl:value-of select="."/> </h2>
</xsl:template>

<xsl:template match="author">
  <li> <xsl:value-of select="."/> </li>
</xsl:template>

<xsl:template match="conf">
  <h3> Conference </h3>
  <p> <xsl:value-of select="."/> </p>
</xsl:template>

</xsl:stylesheet>
```

Figure 1.3: An XSLT program for Web publication. It transforms the XML data trees from Figure 1.1 into the HTML5 data tree from Figure 1.2.

```
$q\langle bibliography\rangle \rightarrow head(title(\text{"}Bibliography\text{"}))\ \cdot\ q\langle child :: *\rangle$
$q\langle article\rangle \rightarrow section(q\langle child :: title\rangle\ \cdot\ h3(\text{"}Authors\text{"})\cdot$
        $ul(q\langle child :: author\rangle\ \cdot\ q\langle child :: conf\rangle))$
$q\langle title\rangle \rightarrow h2(value{-}of(.))$
$q\langle author\rangle \rightarrow li(value{-}of(.))$
$q\langle conf\rangle \rightarrow h3(\text{"}Conference\text{"})\ \cdot\ p(value{-}of(.))$
```

Figure 1.4: Transducer in pseudo code for XSLT program in Figure 1.3.

expression with backwards axis. In other words, the example transducer is working in a top-down and left-to-right manner on the input tree.

Another important point to notice is that both the input and the output tree may be unranked. This means that label of a node does not necessarily determine the number of its children (even without schema specification). Unranked output trees can be constructed by using explicit or implicit concatenation operations when applying a state to all children of the current node in the input tree.

Full XSLT is a much more powerful purely functional programming language, with many more concepts than illustrated by the above example. Most notably, each state may have alongside its argument of type node, a number of parameters of type tree. Furthermore, the output tree can be bound to an XSLT variable, and then used as the input of a follow-up transformation, and this recursively. Thereby, one can define arbitrary n-ary functions from trees to trees, being closed by function composition and recursion.

1.1.3 Transformation Learning

A major drawback of the programming language approach to define transformations of data trees is that programming is inaccessible to non expert users, such as on the Web. This applies to XSLT programs in particular, but equally to other programming languages.

We can, however, try to facilitate the task of specifying tree transformations by designing a system that is able to automatically infer an XSLT program from a given set of examples. This method would have the double advantage to be less tedious, and to require little to no knowledge and practice of XSLT programming. It could also be used to automatically specify fixed transformations as provided by content management systems.

Learning Model

To conceive the inference methods necessary to such a system, we study symbolic machine learning techniques from the area of grammatical inference [de la Higuera, 2010] that started with the work of [Gold, 1967] and [Angluin, 1987]. The domain is a little more general than its name, in that not only grammars may be the target of the inference process, but also automata, transducers, or logical formulas, for words, trees, or graphs. The objective is to identify a machine describing some target language or transformation from a finite sample of positive and negative examples. In the case of language learning, positive examples are words in the target language, while negative examples are words outside of the target language.

In contrast to other learning techniques such as *inductive logic programming* or statistical learning, we wish to learn tree transformations with learning algorithms that satisfy *Gold learning model with polynomial resources* [Gold, 1978] that works as follows: a teacher gives a finite sample of annotated examples for the target machine. In the case of tree transformations, an annotated example is a pair of an input tree and an output tree, with an annotation that says that this pair belongs to the target transformation. The parameter of the problem is a class of tree transformations, to which the target must belong.

A tree transformation can be represented by a pair of an Xml schema, which defines the domain of the transformation, and a subclass of Xslt programs which define how to transform the data trees of the domain. Taking the class of all Xslt programs, however, might be a bad idea, since there does not exist any learning algorithm for this class. It is by far too expressive, even Turing complete [Kepser, 2004, Onder and Bayram, 2006].

So the question is which classes C of tree transformations are well-suited for our purposes. Any such class must respect some important properties. The first property is relevant independently of the learning task, while the two others correspond to Gold learning models with polynomial resources.

L1 The equivalence of two transformations in C is decidable.

L2 The number of annotated examples in a sample needed to characterize a machine of C of size n must be bounded polynomially in n.

L3 The learning algorithm that maps a sample to a machine of C must be in time polynomial in the size of the sample.

Normal Forms

All three points are closely related to the existence of a unique normal form for the machines in the class C. Normal forms such as those we are interested in are usually based on a Myhill-Nerode–like theorem. By Myhill-Nerode–like theorem, we refer to a result that establishes a unique minimal normal form on a class of formal machine, usually by examining the semantics of its states to ensure that each state plays a different, non-trivial role in its machine. For example, such a result exists on deterministic finite automata (Dfa). Given a word language L, two strings x and y are called "L-equivalent" if there is no z such that xz is in L and not yz or the other way around. The Myhill-Nerode theorem [Nerode, 1958] says that this L-equivalence is of finite index if and only if L is regular. Moreover, there exists a unique minimal Dfa recognizing L: the machine where every state is reached by a specific equivalent class. This normal form and the Myhill-Nerode theorem it came from are central to the learning algorithm in polynomial time using a polynomial number of example (Rpni algorithm as seen in [Oncina and Garcia, 1992]).

Learning Algorithms

It is important to note that learning algorithms, or Myhill-Nerode theorems, are rare for word transformation classes, and even more so for tree transformation classes. On words, an important result (see for example [Berstel, 1979, Choffrut, 2003]) is the extension of the Myhill-Nerode theorem to subsequential word transducers, a class of deterministic finite state machines that describe word transformations. This Myhill-Nerode theorem shows that once these subsequential transducers are normalized to produce their output as soon as possible, there exists a unique minimal normal form. This normal form has later been proved to be learnable in polynomial time using a polynomial number of example using OSTIA, a RPNI-like algorithm [Oncina et al., 1993]. While the DFA case had no output to deal with, the question of output normalization is important to learn word transducers. Given a certain input prefix u, the examples have to provide enough information to know the maximum output prefix v we can produce for sure.

1.1.4 Tree Transducers

We next recall the existing classes of tree transducers that are relevant to learning XSLT transformations, with respect to our selection criteria (L1), (L2) and (L3).

More particularly, we will be interested in top-down tree transducers, possibly with macros (these are XSLT's parameters), output concatenation (as we have seen in the example), extended pattern (for XPATH navigation), or regular lookahead. All of these aspects are captured by the navigational macros tree transducers [Maneth et al., 2005] which can even operate on unranked trees, which were proposed as a formal model for the navigational subset of XSLT, for which type checking is decidable. But this class is by far too powerful to consider its learning problem, so we will have to consider subclasses thereof. More generally, we will impose the following two restrictions.

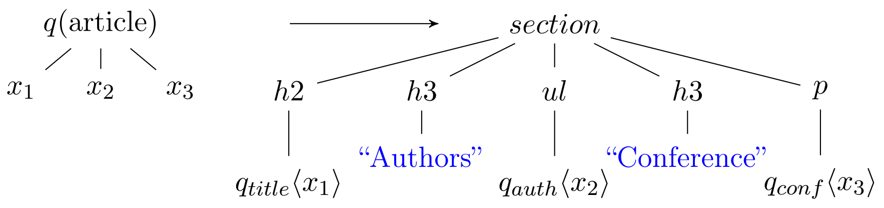
R1 We will consider finite signatures only, so we ignore the infiniteness that may come with data values.

R2 We will consider ranked trees only, so we ignore aspects of unrankedness. These aspects can be dealt with via ranked encodings of unranked trees.

Macro Tree Transducers (MTTs) We consider traditional MTTs [Engelfriet and Vogler, 1985], a very expressive class of top-down tree transducers on ranked trees. As we previously mentioned, their specificity comes from the fact that states can remember a finite number of parameters, i.e. of output subtrees built from transforming other parts of the input tree, to be used in the transformation of a possibly distant part of the input.

This expressiveness, however, comes at a huge computational cost: the equivalence problem is a long-standing open question on this class. One of the only results found in that domain is decidability of a fragment of this class [Engelfriet and Maneth, 2003, Engelfriet and Maneth, 2005], through a non-elementary translation into Mso-definable transformations, and the decidability of (deterministic) top-down tree-to-word transducers with concatenation [Seidl et al., 2015] which can be simulated by Mtts.

Deterministic Top-Down Tree Transducers (DTops). A better candidate is the less expressive class of DTops. It is a well-known and largely-studied class of tree transducers for which equivalence is known to be decidable [Ésik, 1980]. A DTop is a finite state machine that rewrites a tree from the root to its leaves. For example, a DTop that could compute the Xml to Html transformation presented above could contain a rule as described in Figure 1.5. Each leaf of the form $q'\langle x_i\rangle$ marks a call to state q' on the i^{th} son



q(article)
x_1
x_2
x_3
$section$
$h2$
$h3$
ul
$h3$
p
$q_{title}\langle x_1\rangle$
"Authors"
$q_{auth}\langle x_2\rangle$
"Conference"
$q_{conf}\langle x_3\rangle$

Figure 1.5: Rule reading a article tag.

of the node we just read. We note that DTops possess the capacity to reorder subtrees, to copy subtrees (by having two pairs $q'\langle x_i\rangle$ in the same rule) or to delete subtrees (by having no pair $q'\langle x_i\rangle$ in some rule).

This class allows to model some basic tree transformations, and most of the results we had in word transducers and tree automata can be extended to DTops. Engelfriet, Maneth and Seidl [Engelfriet et al., 2009] showed that DTops have a unique normal form. This result was later extended in [Lemay et al., 2010] in several relevant ways. The first one was the proof of a Myhill-Nerode theorem, based on a notion of top-down origin: for every output node in a production of a DTop, one can pinpoint a unique input node as being "responsible" for its existence. This Myhill-Nerode theorem leads to the same normal form as the one of [Engelfriet et al., 2009], but also to a Rpni-like algorithm to learn this normal form in polynomial time using a polynomial number of examples.

Extended Tree Transducers. The class of extended tree transducers is tailored to allow for pattern-matching in the rules instead of the simple top-down rewriting of DTops. This class emulates part of Xslt's ability to do tests on subtrees, retrieving information from a few levels below the node it currently reads, and adapt its production accordingly. For example, a rule that checks the contents of < conf > at the moment it reads a tag < article > can be seen in Figure 1.6.

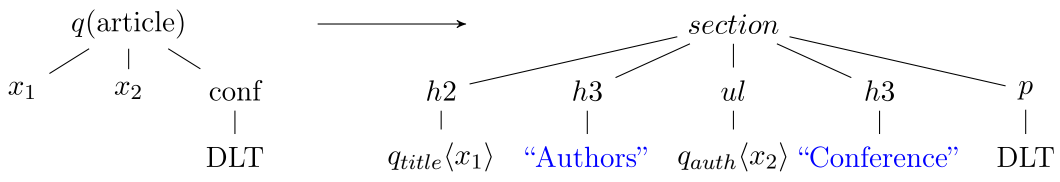


Figure 1.6: Rule pattern-matching a DLT article.

Very few results exist on this class for the problems we consider. Their expressive power was studied in [Maletti et al., 2009] with applications to the field of natural language processing in mind. For our purposes it is relevant to note that extended top-down tree transducers are strictly more expressive than DTops, but strictly less expressive than DTops with regular lookahead [Engelfriet, 1977]. This notably means that equivalence of extended tree transducers is decidable (see [Maneth, 2015]). One particularity of this class is that it is not stable by composition. This property is detrimental to its relevance, and was addressed in papers that try to find fragments to regain this property: [Fülöp and Maletti, 2013] consider for example the case of linear ε-free extended transducer, i.e. where the input trees cannot be copied several times in the output, and where every rule must consume at least one input symbol. For this class, the composition hierarchy eventually collapses. Another result [Benedikt et al., 2013] more directly linked to static analysis of tree transformations, shows that compositions of functional extended linear top-down tree transducers with regular lookahead describe a class of transformations where determinacy is decidable. The problem of determinacy is to decide if, given two transformations, the output of the first transformation still contains enough information to compute the second transformation.

DTops with regular lookahead. Another, more general class of tree transducers that allows for patter-matching is the class of DTops with regular lookahead [Engelfriet, 1977]. Regular look-ahead means that the DTop comes with a total deterministic bottom-up automaton, that labels every node of the tree with its states. This information can then be used by a DTop whose input signature is the cartesian product of input symbols and lookahead states. This can be seen as the possibility to perform a regular test on the subtrees of a node at every step of the top-down reading of the input tree.

To study this class, it is useful to consider the case of lookahead transducers on words first. Lookahead word transducers (DWT$_\ell$) combine a total right-to-left labeling DFA with a left-to-right subsequential word transducer. This class has been studied and provided with multiple interesting results. Its equivalence is decidable by reduction to the subsequential case [Berstel, 1979]. There is also a Myhill-Nerode theorem [Reutenauer and Schützenberger, 1991] that defines a normal form on DWT$_\ell$ using a lookahead with a minimal number of states.

For DTOPs with lookahead, results are more scarce: while the equivalence problem is still decidable, by a reduction to the DTOP case (see for example [Maneth, 2015]), there exists no known normal form on trees. The problem in known to be difficult and remains open.

Tree to Word Transducers. Most classes of top-down transducers lack the possibility to combine two hedges into a bigger hedge in their output, even with a First Child/Next Sibling encoding, that plays an important role in XSLT. To model this concatenation power with tree transducers is a highly challenging problem. A reasonable approach would be to first study the consequences of concatenation in the output for the simpler case where the output is a word, rather than a tree or hedge.

The class of tree-to-word transducers is of particular relevance, as it can be seen as a tool to model the very particular case where each tree in the hedge produced is a single symbol. They are notably stable under concatenation. The study of tree-to-word transducers is also relevant as it can be seen as a restricted case of the previously mentioned Macro-tree transducers [Engelfriet and Vogler, 1985]. In fact, if we consider Macro-tree transducers on an unary output signature, they would encompass tree-to-word transducers.

This new concatenation power, however, appears to be difficult to combine with the classical techniques of language theory, which leads to few results on this tree-to-word transducers as a whole. Equivalence for all tree-to-word transducers has recently been proved to be decidable [Seidl et al., 2015] with a co-randomized polynomial algorithm for the linear case. Note that this result uses neither classic logic methods, nor the classic transducer methods, and does not provide a characterization nor a Myhill-Nerode theorem. Rather, this result is first proved on transducers with a unary output alphabet, using classical fixpoint results. This result is then non-trivially extended to the general case, using polynomial ideals and Hilbert's basis theorem.

There exists, however, one particular fragment of the class of tree-to-word transducers that has been studied, and provided with several results that successfully extends classic transducer methods to transducers with computation in the output. This is the class of sequential tree-to-word transducers (or STWs), that prevent copying in the output and force subtrees to produce following the order of the input.

On this fragment, a Myhill-Nerode result has been proved [Laurence et al., 2011]. It yields a unique minimal normal form on STWs, that has later been proved to be learnable in PTIME, using a RPNI-like algorithm [Laurence et al., 2014]. Uncharacteristically, the equivalence problem has been proved to be polynomial [Laurence et al., 2011], not by computing and comparing minimal normal forms (which would give an algorithm in exponential time) but as a reduction to the problem of morphism equivalence on Context-Free Grammars [Plandowski, 1995].

1.2 Limitations of Transducer Learning

The XML technology mainly serves for transforming data trees from one schema to another. Such transformations are typically defined by an XSLT program and an XML schema for restricting its application domain. Whether all trees in the range of the transformation then satisfy the expected output schema is an orthogonal type checking question, which is known to be decidable for large classes of such transformations as long as the schemas define regular tree language [Maneth et al., 2005], thus excluding constraints on data values. In what follows we consider subclasses of XSLT programs that can be modeled by transducers, while focusing on learning aspects.

We now present three ways in which the existing learning algorithms for classes of transducers are limited.

1.2.1 Schema Restrictions

The first limitation is that the learnability of a class of transducers is not necessarily preserved by schema restrictions. Given a schema S and a transducer M, let the pair (M, S) define the same transformation than M but with the domain restricted to the language of S. Given a fixed schema S and a class of transducers C that is learnable in Gold's model with polynomial resources, the question is then whether the class of transformations defined by transducer-schema-pairs

$$C_S = \{(M, S) \mid M \in C\}$$

is learnable too. If the schema S can be checked by transducers in the class of C itself, meaning that the transformation of (M, S) can always be defined by some transducer in C, one can often adapt the learning algorithm for C to lo learn C_S. For instance, one can learn subsequential transducers on words with regular schema restrictions, since subsequential transducers can integrate regular schema restrictions [Oncina and Varo, 1996]. Similarly, regular schema restrictions can be integrated into the learning of tree automata [Niehren et al., 2013].

However, non-regular schema restrictions cannot always be integrated into subsequential transducers. The same problem arises already for the learning of regular languages defined by DFAs. Consider a schema S that defines the non regular language $[\![S]\!] = \{a^n b^n \mid n \in \mathbb{N}\}$. The question is whether the class $\text{DFA}_S = \{(A, S) \mid A \text{ a DFA}\}$ is are learnable. One could try to use the learning algorithm RPNI for DFA. But the problem is which normal forms to use for DFA_S. Consider the two regular languages $\{a, b\}^*$ or a^*b^*. Their intersections with $[\![S]\!]$ are equal, so we could choose either the unique minimal DFA of the one or the other as a normal form. But which one to prefer? Even worse, we may not be able to choose either of them since not all positive examples that are needed to learn these DFAs may belong to $[\![S]\!]$.

The limitation is particularly relevant for the class of DTOPs, even when choosing top-down schema restrictions where the schemas are top-down deterministic tree automata. As noticed by [Engelfriet et al., 2009], DTOPs cannot always integrate top-down schema restrictions, which makes their normalization considerably more complicated than for subsequential transducers. The problem has been solved by considering DTOPs with top-down domain inspection, that is the class DTOPI_{td} of pairs of DTOPs and top-down deterministic tree automata. In the same line [Lemay et al., 2010], a Myhill-Nerode theorem and a learning algorithm were obtained for the class DTOPI_{td}. But whether classes of DTOPs with more expressive schema restrictions can be learned remains open.

1.2.2 Lookahead

The second problem is that the learnability of a class of transducers may not be preserved when adding lookaheads. A lookahead allows to annotate the nodes of a tree (or the positions of a word) with extra information, before the transducer is run. Equivalently, one can enhance a transducer to look ahead into the remainder of a tree, that it would see only in the future otherwise.

It should be noticed that adding lookaheads to a class of transducers does not affect the decidability of the equivalence problem of that class. For example, the equivalence problem of DTOPs with regular lookahead is decidable, as it can be reduced to the equivalence problem of DTOPs, since the lookaheads of two DTOPs can be synchronized.

However, when it comes to transducer normalization and learning, the addition of regular lookahead is problematic. To normalize transducers with lookahead, a natural idea is to minimize the lookahead in a first step, and then the transducer in a second. This can indeed be done for deterministic transducers on words with regular lookahead as shown in

[Reutenauer and Schützenberger, 1991]. These machines capture the well-studied class of rational functions on words. However, the problem of whether rational functions on words can be learned is a long open standing, mainly since the normal form obtained by lookahead minimization is difficult to exploit in a learning algorithm.

The situation for DTOPs with regular lookahead is even more complicated. It is open whether the lookahead of such machines can be minimized. The most recent result in this domain [Engelfriet et al., 2016] works for the very restricted class of DTOP with lookahead that are total, deterministic, ultra-linear (i.e. with no cycle than can copy subtrees), and bounded erasing (i.e. with no cycle than can erase subtrees).

1.2.3 Output Concatenation

The third problem is whether transducers with concatenation operations in the output can be learned. For instance, one can extend DTOPs so that they output sequences of trees which can be concatenated, similarly to what was proposed for transducers in the context of XSLT [Maneth and Neven, 1999, Maneth et al., 2005] .

The simplest class is that of word transducers with output concatenation [Filiot and Reynier, 2014], which was then generalized to tree-to-word transducers. Only very recently it could be shown that the equivalence problem of such tree-to-word transducers is decidable [Seidl et al., 2015]. But the proof doesn't rely on a normal form, so one cannot hope to lift this approach to a learning algorithm.

The only existing normalization and learning results are restricted to sequential tree-to-word transducers [Laurence et al., 2011, Laurence et al., 2014], i.e., linear transducers that rewrite their arguments from the left to the right. Furthermore, the equivalence problem for sequential tree-to-word transducers is in polynomial time. Whether larger classes of tree-to-word transducers can be normalized, or can be checked for equivalence efficiently is open. Note that since Macro-tree transducers on an unary output signature encompass tree-to-word transducers, any result furthering our understanding of tree-to-word transducers can be seen as a necessary step forward to obtain similar results in Macro-tree-transducers.

1.3 Contributions

Our three main contributions tackle aspects of the above three limitations of transducer learning.

1.3.1 Learning DTops with Regular Inspection

We show that DTops with regular domain inspection (DTopI$_{\text{reg}}$) can be learned in a Gold-style model with polynomial resources. The regular domain of the target transformation is a parameter of the learning problem. The polynomial complexity, however, do not directly depend on the size of the target DTopI$_{\text{reg}}$. Rather, it is based on the number of semantic alignments, a notion we introduce to get Myhill-Nerode–like equivalence classes. That number may be exponential in the size of the target DTop in the worst case.

Our learning result extends the one for DTops with top-down inspection (DTopI$_{\text{td}}$) from Lemay, Niehren, and Maneth [Lemay et al., 2010], which in turn is based on a normal form for these transducers from Engelfriet, Maneth, and Seidl [Engelfriet et al., 2009]. In this case, the normal form of a DTopI$_{\text{td}}$ ensures that the DTop works in synchronization with the top-down deterministic tree automaton recognizing its domain. This is no longer possible for regular domains, as these cannot always be defined by top-down deterministic tree automata (while bottom-up determinism would be sufficient).

The extension to regular schemas is relevant for two main reasons. First, it allows to consider DTops with restrictions by RelaxNG schemas, since these can express all regular tree languages (in contrast to Xml Schema). This higher expressiveness is appreciated in applications such as Sbml, whose earlier versions were defined by an Xml Schema, and whose current version is defined by an RelaxNG schema. Furthermore; the ranges of Web publishing transformations are often regular, but not top-down. As an example, let us consider a Web publication task that sends Xml files following a top-down schema to Html files belonging to a schema that has a data constraint (see Figure 1.7), because it copies an information from the input in two different leaves of the output.

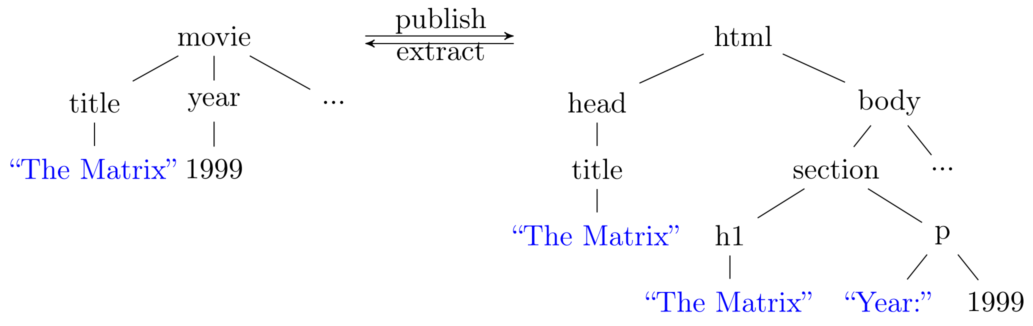


Figure 1.7: Web production that creates some data equality constraint in its output.

Since the title is copied twice, the range of this transformation is not top-down. It may be even non-regular if the number of possible titles is infinite. However if we consider this transformation on a finite number of titles, this range is regular. When interested in learning the inverse transformation as needed for information extraction, we thus have to learn a $\text{DTopI}_{\text{reg}}$.

Let us consider the problem of learning such a transformation and the $\text{DTopI}_{\text{reg}}$ that describe it. The new difficulty compared to learning a DTopI_{td} comes from the redundancy introduced by Web publishing, that leads the schema not to be top-down anymore. The previous learning algorithm for DTopI_{td} heavily relies on the fact that each output node must have a unique origin in the input tree, which is the only node of the input tree which may be "responsible" for its existence. However, in the presence of redundancy, such origins might no longer be unique, i.e. it becomes possible that an information in the output tree may be deduced from different sources in the input. In our example of Web information extraction, which is the inversion of the Web publication transformation form Figure 1.7, the encoded title of the movie can be inferred from the title of the page, or in the $h1$ tag in the section of the body. This loss of uniqueness is at odd with the usual obtention of a unique normal form by a Myhill-Nerode style theorem.

To this end, we introduce the notion of syntactic and semantic alignment, linking input paths and output paths of the transformations we study. Syntactic alignment aims to describe which part of the input is read by a DTop to produce which part of the output. It is, as its name implies, a purely syntactic property, describing the functioning of the transducer. Semantic alignments, on the other end, are a property of the DTop's transformation: it describes which part of the input contains the information necessary to produce which part of the output. Both these notions can be seen as a reformulation of the notion of io-paths presented in [Lemay et al., 2010]. In earliest DTops with top-down inspection, both those notions coincide. This ensures the existence of a Myhill-Nerode theorem, as there is at most as many different kinds of semantically aligned pairs as there are states in the earliest DTop.

We cannot find a Myhill-Nerode in the same way for regular inspections: some semantically aligned pairs are not visited by some DTops. We extend the Myhill-Nerode theorem to the $\text{DTopI}_{\text{reg}}$ class by proving that regular schemas do not introduce an infinite number of new semantically aligned pairs.

Regular schemas also present a challenge to define a clear unique normal form: because of the possible redundancy, some transducers have a choice as to where they find the information necessary to produce their output. To define a normal form on this class, we further impose a choice on the source of the output when redundancy leaves the choice open.

We also wish to extend the learning algorithm of [Lemay et al., 2010] for DTops with regular inspection. For DTops with top-down inspection, since all semantically aligned pairs are used in the normal form, we use a Rpni-like algorithm to find all the minimal representatives of the finite number of equivalence classes of semantically aligned pairs. Each of these representatives will have its state in the normal form we learn.

The fact that for a DTopI$_{\text{reg}}$ some semantically aligned pairs are unused makes this method insufficient. We have to find out which semantically aligned pairs will be visited by the normal form, and which ones will not. We solve this problem by exploring semantically aligned pairs with a Rpni-like algorithm, from the smallest to the longest, until we find enough equivalence classes to build a DTopI$_{\text{reg}}$ in normal form.

We note that while regular schema restriction is an improvement over DTops with top-down schema, it cannot encode the full scope of what Xml schema could encode: since data trees are not considered by the DTop formalism, all data constraints are but encoded in a finite signature. These encodings are likely to often go beyond the scope of regular languages. The problem of schema restriction remains open for non-regular languages, and as long as it is not addressed in the general case, each new class we consider is liable to require a new extension of both a Myhill-Nerode result and a learning algorithm.

1.3.2 Learning Rational Functions

We propose a learning algorithm for deterministic subsequential word transducers with lookahead (Dwt$_\ell$). This class corresponds to all rational functions on words [Elgot and Mezei, 1965]. Not only was learning rational functions an open problem, it is also to our knowledge a rare occurrence to learn a class of transducers with lookahead.

The main difficulty we need to circumvent to learn the normal form of [Reutenauer and Schützenberger, 1991] is that its equivalence class is hard to learn with a finite number of examples: two suffixes are equivalent, and can be sent to the came state of the lookahead, if replacing one with the other at the end of a word only creates a bounded difference in the output.

To palliate with this problem we say that there is a bound m such that if two suffixes creates more than m differences in the output, then they create arbitrarily big differences in the output.

While this bound exists for all rational functions, it is not easily known or learned. If we are to underestimate this bound, then we would learn a lookahead finer than the one of the normal form of [Reutenauer and Schützenberger, 1991].

The solution we propose is a to learn a normal form that does not necessarily have a minimal lookahead, but that can be learned by guessing the bound m, the number of states n the Dwt$_\ell$ would need, then trying to build a Dwt$_\ell$ under n states while assuming any suffixes creating more than m output differences are not equivalent. If it fails, the algorithm tries again with a bigger bounds m and n.

The result of this algorithm might not have a minimal lookahead, if it manages to find a Dwt with a reasonable number of states to complete its lookahead. When compared to the normal form of [Reutenauer and Schützenberger, 1991], it strikes a bargain between the roles of the lookahead and of the Dwt.

Whether the normal form of [Reutenauer and Schützenberger, 1991] can actually be learned or not is unknown: from our normal form, finding a Dwt$_\ell$ with minimal lookahead is tantamount to lookahead minimization. As previously mentioned, even deciding if a lookahead can be completely removed is a challenging problem. We also note that the problem of finding a normal form for DTops with lookahead remains open: the techniques of this paper do not translate easily into trees.

1.3.3 Linear Tree to Word Transducers

We present Linear tree-to-word transducers (Ltws), whose only constraint is to not create copies of an input subtree. On this class, we provide a Myhill-Nerode algorithm, leading to an earliest minimal normal form. Furthermore, the equivalence problem on these linear transducers remains decidable in polynomial time, through a non-trivial reduction to the Stw case. The lift of the restriction forbidding reorderings in Stws is a necessary step to study tree-to-word transducers, and hope to find a normal form on this class.

The possibility of reordering subtrees lead to a complication for normalization of Ltws, as two transducers can be equivalent but use different reorderings. The notion of earliest transducers can be directly imported from Stws, but it is no longer enough to define a normal form.

To fix this issue we show that two earliest Ltws can be equivalent while using different reorderings if and only if they exclusively reorder periodic productions of same period. By then imposing an order on subtrees whenever a periodic production leaves the choice open, we provide an ordered earliest normal form for Ltws.

The adaptation of the polynomial equivalence test proves to be more challenging. In the Stw case, the strategy is to reduce the problem to the morphism equivalence problem on a context-free grammar [Plandowski, 1995]. This means that the fact that both transducers use the same reordering is central to the reduction.

The characterization of possible reorderings we established is only made for earliest Ltws, and making a Ltw earliest can only be made in exponential time. Hence the computation of the normal form or even of a fully earliest Ltw is out of the question to preserve a Ptime algorithm.

The solution we propose is to only compute the earliest form for the few states that can possibly be reordered, i.e. those whose production can become periodic in an earliest Ltw. These states are both easier to make earliest, using word combinatorics results, and the only relevant one to reduce Ltw equivalence to STW equivalence. We finally prove this reduction to be in PTIME, which leads to a PTIME algorithm to decide LTW equivalence.

We do not present the learning algorithm for the normal form we provide on LTWS. It would likely involve an adaptation of the learning algorithm on STWS [Laurence et al., 2014], but requires additionally to learn in polynomial time in what order the subtrees' images should be sorted. The extension on the normal form to the general tree-to-word transducers case remains open, and is likely to be complicated. The main challenge is likely to reside in identifying which fragment of the output comes from which subtree in the input.

Outline

After some related work on tree transformations presented in Chapter 2, Chapter 3 reminds some preliminary properties on word, trees, their automata, and presents the RPNI algorithm for DFAS.

Chapter 4 and 5 study DTOPS with top-down and regular inspection. Chapter 4 elaborates on the results of [Lemay et al., 2010]. It extends the definition of earliest and compatible to regular inspection, and presents the Myhill-Nerode theorem, normal form, and learning algorithm on DTOPS with top-down inspection. Chapter 5 builds on the definition of earliest and compatible of Chapter 4 to prove a new Myhill-Nerode theorem, normal form, and learning algorithm on DTOPS with regular inspection.

Chapter 6 studies DWT$_\ell$. It proposes a learning algorithm in polynomial time and data for rational word transformations that the class of DWT$_\ell$ describes.

Finally, Chapter 7 and 8 presents the class of linear tree-to-word transducers (LTWS), a fragment of the tree-to-word transducers class that forbids copies of its input. In Chapter 7 we characterize how a LTW can reorder the images of its subtrees without changing its output, and establish a normal form on LTWS, called the earliest ordered normal form. Chapter 8 proves that equivalent is decidable in polynomial time for LTWS, by reduction to the equivalence problem on sequential tree-to-words transducers.

Publications

Some of the results presented in this thesis were previously published in international conferences and workshops. We list here those publications.

Chapter 5 is an extended version of the paper *Learning Top-Down Tree Transducers with Regular Domain Inspection* published in ICGI 2016 [Boiret et al., 2016b].

Chapter 6 corresponds to the paper *Learning Rational Functions* published in DLT 2012 [Boiret et al., 2012].

Chapter 7 corresponds to the paper *Normal Form on Linear Tree-to-Word Transducers* published in LATA 2016 [Boiret, 2016].

Chapter 8 corresponds to the paper *corresponds to the paper* published in DLT 2016 [Boiret and Palenta, 2016].

Beyond the results discussed in this document, I worked during the time of my PhD project on the CoLiS project[1] on the verification of Linux installation scipts. The idea is to model such scripts as tree transducers that change the file system tree.

In *Deterministic Automata for Unordered Trees* published at the special isssue of GandALF 2014 [Boiret et al., 2014] in the Journal of Information & Computation 2016 [Boiret et al., 2016a], we present a framework to describe classes of deterministic bottom-up tree automata, parametrized by the tests that explore a node's arity in a bottom-up rule. We then propose three such classes of various expressivity, the most general defining the same languages as Mso logic formulae with Presburger tests on arities, and the least general as Mso logic formulae with counting constraints.

In *Logics for Unordered Trees with Data Constraints* presented at Lata 2015 [Boiret et al., 2015], we study Mso logics on unordered data trees with counting constraints and comparisons of data values of siblings. We prove the satisfiabilty problem in this logic to be decidable.

[1] ANR Project CoLiS, Correctness of Linux Scripts, http://colis.irif.univ-paris-diderot.fr/

CHAPTER 2

Related Work on Transducers

We present some existing work on equivalence, normal form, and learning problems on word and tree transformations. We give some particular focus to the class of MSO defined transformations, as they provide a wide expressive reference point with interesting decidability results.

Contents

2.1 MSO Transformations 21
2.1.1 Words 22
2.1.2 Trees 23
2.2 Bottom-Up Transducers 24
2.3 Transducers for Data Trees 25

2.1 MSO Transformations

If tree transducers can be used to describe and study tree transformations, there exists results that come from the study of tree transformations as described by logics. By far the most studied logic in this regard has been the MSO logic [Courcelle, 1994, Engelfriet and Courcelle, 2012].

The MSO logic is known to have numerous links to formal languages. The domain of word and tree transformations is no exception. MSO formulae can be used to describe a quite vast class of transformations, usually as much as some of the most expressive transducer classes in words or trees. Furthermore, since MSO satisfiability is decidable, most of those classes inherit decidability results for equivalence.

We consider MSO formulae on graphs labeled on a finite alphabet. These formulae can contain judgements on node labels, i.e. if x is a node, and a a label, then $l_a(x)$ is true if x is labeled by a. These formulae can also contain judgements on edges, i.e. if x, y are nodes, then $\theta(x, y)$ is true if there is an edge from x to y. As an example, a formula verified by a complete graph would be:

$$\forall x \forall y \; \theta(x, y)$$

MSO formulae on words or trees can be considered as the particular case where $\theta(x, y)$ represent the relation from one letter to the newt in words, or the parent-child relations in a tree.

Closed MSO formulae can be seen as a way to describe languages. As an example, the formula we described above would describe the language of all complete graphs. MSO languages have interesting properties: in graphs, it is equivalent to graph languages edscribable by Hyperedge Replacement or by Vertex Replacement. In trees, MSO languages correspond to regular tree languages, i.e. tree languages described by tree automata. Similarily, in words, MSO languages correspond to regular word languages, i.e. word languages described by word automata.

To describe transformations using MSO formulae, we no longer consider a judgement on graphs but a graph and n copies of this graph. The image is constituted by the n copies, linked by edges described by judgements of form $\theta_{i,j}(x, y)$, which says that the i^{th} copy of x is linked to the j^{th} copy of y. As an example on words, imagine we want to transform words u into a concatenation of themselves with their mirror image $u\overline{u}$. We consider a MSO formula using two copies: the first one will copy u exactly, and the second one will represent the mirror image of u:

$$\forall x, y\, \theta(x, y) \Leftrightarrow \theta_{1,1}(x, y) \wedge \theta(x, y) \Leftrightarrow \theta_{2,2}(y, x)$$

We then ensure that the first copy is concatenated with the second copy by linking them by the last letter of u:

$$\forall x, y\, \theta_{1,2}(x, y) \Leftrightarrow [x = y\ \wedge \neg\exists z \theta(x, z)]$$

One of the most important results on MSO transformations for the problems we study is that while the equivalence problem is undecidable on graphs [Courcelle, 1994], it is decidable for deterministic MSO graph to words or graph to trees transformations [Engelfriet and Maneth, 2005]. This result is vital, as it implies that equivalence is decidable for deterministic MSO words or trees transformations. Those deterministic MSO transformations are quite expressive, and this result is one of the largest equivalence decidability result both for word or tree transformations.

2.1.1 Words

MSO transformations on words have several interesting properties. The type-checking problem, which is to see if the inverse image of a regular language by a certain MSO transformation is included in another regular language, is decidable [Courcelle, 1994]. There are also two formalisms of word transducers that are exactly as expressive as MSO transformations: two-ways word transducers [Engelfriet and Hoogeboom, 2001], and streaming transducers.

Two-way word transducers can be seen as the natural extension of two-way automata to create an output, which in turn makes their equivalence to Mso transfiormation an elegant extension of the fact that two-way automata describe regular languages, which is also the class of Mso-definable word languages. No normal form is known on this formalism. The closest result towards this goal is [Bojańczyk, 2014], that asserts that if a two-way transducer also provides information to explicit which input letter was read to produce each output letter, then this new class of transducers "with Origin Information" has a Myhill-Nerode theorem and a normal form. It is worth noting, however, that deducing origin information without the transducer explicitely making so remains a challenging open problem.

Another equivalent formalism is streaming transducers [Alur and Cerný, 2011]. These transducer build an image in one pass, by storing multiple output variables that it can copy, combine, or increment at each step. Their advantages are that they are better suited for streaming purposes, as contrary to Mso or two-way transducers, their image is built during one single pass. Their equivalence and type-checking problems are in Pspace. They have been proven to be just as expressive as Mso transformations and two-way tree transducers [Alur and Cerný, 2010].

2.1.2 Trees

The comparison to Mso tree transformations to the diverse classes of tree transducers considered in the state of the art is not quite as direct as in the word case due to linear size increase. While Mso tree transformations are built upon a bounded number of copies of the input, tree transducers are quite quickly capable to produce an arbitrarily great number of copies of some subtrees to compute an output. One such example would be a DTop that transforms a unary tree into a complete binary tree:

$$\begin{aligned} q(a(x_1)) &\to A(q\langle x_1\rangle, q\langle x_1\rangle) \\ q(\#) &\to \# \end{aligned}$$

Since the image of an input tree with n nodes is a binary tree of size 2^n, there is no bound k such that for any input tree, the output can be built from k copies.

This capacity to copy some subtrees an arbitrarily great number of time is not particularily desirable in most applications. Particularily, in document processing, it stands to reason that any information in the input ought to appear a bounded number of time in the output document. In most cases wwe can even assume this bounded number to be 1.

For this reason, many classes of transducers are studied in the particular restricted case of *linear size increase* (or *with bounded copies*). The condition correspond to the case where a transducer is limited in the number of copies it can make of a subtree, usually by forbiding a state cycle using rules that create copies. Our previous example does not satisfy this restriction, as the rule $q(a(x_1)) \to A(q\langle x_1\rangle, q\langle x_1\rangle$ allows q to loop on itself while creating copies, which leads to the transducers producing an unbounded number of copies of its lower leaf.

Under the linear size increase constraint, MSO transformations are once again a gold standard of expressivity. It is remarkably as expressive as macro tree transducers with bounded copies [Engelfriet and Maneth, 2003, Engelfriet and Maneth, 2005]. This result is of particular relevance as it allows any transducer class with linear size increase which are proven to be less expressive than MSO transformations or macro tree transducers (that is to say virtually any reasonable transducer class) can inherit the principal results of MSO tree transformations. The two major results for our considerations are the decidability of the equivalence problem [Engelfriet and Maneth, 2005] as well as the decidability of type-checking [Maneth et al., 2005].

2.2 Bottom-Up Transducers

While most transducer classes considered work from top to bottom, some transducer classes are bottom-up in their way to compute an image. We briefly present some result relevant to the problem we study on these classes.

The class of node-selecting tree transducers [Carme et al., 2004] is a class of relabeling bottom-up transducer designed to annotate some nodes of the input tree. The goal is to select exactly all nodes selected by some XPATH query. While these transducers do noot modify the structure of a tree in anyway, this class possess one of the only learning algorithms on tree transducers prior to [Lemay et al., 2010]: there exist a normal form and a learning algorithm in polynomial time and data for this class.

The class of deterministic bottom-up tree transducers have been studied, and compared to their top-down counterpart [Engelfriet, 1975]. Unlike the automata case where the bottom-up formalism is strictly more expressive than their top-down counterparts, the bottom-up and top-down formalisms of deterministic tree transducers are incomparable. It is worth noting that both classes are less expressive than DTOPs with a regular lookahead. The class of bottom-up tree transducers possess a minimal normal form [Friese et al., 2010], as well as an algorithm to decide the equivalence problem [Friese, 2011]. We note that this normal form has yet to grant a learning algorithm for the class of bottom-up tree transducers.

2.3 Transducers for Data Trees

Most classes of transducers we presented model XSLT's ability to read, test and transform a tree's structure, but very few concern themselves on data-centric question. An XSLT program can do some tests on the data contained in the input tree, and copy or modify them when producing its output.

These abilities are both difficult to model, as it forces to forsake the advantages of working on finite input and output signatures, but are also known to be difficult to deal with for static analysis questions (see for example [Bojanczyk et al., 2011]).

Symbolic automata and transducers [Veanes et al., 2012, Veanes and Bjørner, 2011] work on data words and trees, where each letter or node is labeled by some data. These classes are defined modulo a background theory that defines what tests and operations can be performed on the labeling data. For example, if our nodes are annotated by integers, the Presburger theory defines Presburger symbolic automata and transducers.

One of the major draws of this class is that they work on a possibly infinite signature while still offering the possibility of using classical formal languages methods, as long as the background theory. Notably, [D'Antoni and Veanes, 2014] presents a minimal normal form for symbolic word automata. On word transducers, [Veanes et al., 2012] proves that under the condition of background theory decidability, there exists algorithms to decide the equivalence of symbolic word transducers. While this result does not translate entirely into tree transducers, [Veanes et al., 2012] provides a decision algorithm for the equivalence of symbolic tree transducers under the condition that they are linear (no rule can use the same subtree twice).

Chapter 3 Preliminaries

Abstract. We recall the definitions of finite automata for words and trees, discuss their Myhill-Nerode theorems, and illustrate how they can be lifted to automata learning algorithms in Rpni style in the case of Dfas.

Contents

3.1 Words . . . 27
3.1.1 Word Automata . . . 28
3.1.2 Determinism . . . 29
3.1.3 Myhill-Nerode Theorem . . . 30
3.2 Trees . . . 31
3.2.1 Tree Automata . . . 32
3.2.2 Determinism . . . 34
3.2.3 Myhill-Nerode Theorem . . . 35
3.3 Automata Learning . . . 36
3.3.1 Learning Model . . . 37
3.3.2 Characteristic Samples . . . 38
3.3.3 Learning Algorithm Rpni . . . 40

3.1 Words

In this thesis we will consider finite words on a finite alphabet.

An *alphabet* Σ is a finite set of Symbols. We denote by Σ^* the set of finite words over Σ with the concatenation operator $\cdot$ and the empty word ε. Note that this operator is often implied, making uv another notation for $u \cdot v$. For a word u, $|u|$ is its length. We call language a set of words $L \subseteq \Sigma^*$.

For $w = u \cdot v$, the left quotient of w by u is $u^{-1} \cdot w = v$, and the right quotient of w by v is $w \cdot v^{-1} = u$. We extend those notions to languages: $u^{-1}L = \{v \mid uv \in L\}$ For a language L, we denote $lcp(L)$ the longest word u that is a prefix of every word in L, or *largest common prefix*. Also, $lcs(L)$ is the largest common suffix of L.

3.1.1 Word Automata

We recall the notion of word automata, their notion of determinism, and the Myhill-Nerode theorem that gives a unique minimal normal form.

Definition 1. *A* (nondeterministic) word automaton *(usually called* NFA*) is a quadruple* $A = (\Sigma, Q, Q_I, Q_F, rul)$ *composed of*

- *a finite alphabet* Σ,
- *a finite set* Q *of states,*
- *a subset* $Q_I \subseteq Q$ *of initial states,*
- *a subset* $Q_F \subseteq Q$ *of final states,*
- *a binary relation* $rul \subseteq (Q \times \Sigma) \times Q$ *of transitions.*

We usually denote a rule (q, f, q') as $q \xrightarrow{f} q'$. For any states $q \in Q$ we define membership to the language $[\![A]\!]_q$ accepted by A when started at q by induction on the words, such that:
- if $q \in Q_F$, then $\varepsilon \in [\![A]\!]_q$
- if $(q, f, q') \in rul$, and $u \in [\![A]\!]_{q'}$, then $fu \in [\![A]\!]_q$

We define the language that A accepts by $[\![A]\!] = \cup_{q \in Q_I} [\![A]\!]_q$. A word language L is called *regular* if $L = [\![A]\!]$ for some NFA A.

To put the classical results of NFAs in parallel with similar results on tree automata and then tree transducers, we will present briefly a few definitions and properties of residual languages $u^{-1}L$ of regular languages L.

For any NFA A and word u on the alphabet of A, we define the set of states *reached by* u *in* A as the set $Q_u(A)$ that A may reach after reading u as follows by induction on the length of u:

$$\begin{array}{rcll} Q_\varepsilon(A) & = & Q_I, & \text{(the initial states of } A) \\ Q_{uf}(A) & = & \{q' \in Q \mid q \xrightarrow{f} q',\ q \in Q_u(A)\}. & \end{array}$$

Definition 2. *A* NFA $A = (\Sigma, Q, Q_I, Q_F, rul)$ *is* trimmed *if for all* $q \in Q$:

co-reachability: $[\![A]\!]_q \neq \emptyset$, *and*

reachability: *there exists a word* $w \in [\![A]\!]$ *and a word* u *such that* $w = uv$ *and* $q \in Q_u(A)$.

Every NFA can be made trimmed in linear time by removing all useless states, i.e., those that cannot be reached from an initial state or cannot reach a final state. Therefore, we can assume that all NFAs are trimmed whenever this will be needed in what follows.

We next state a simple but fundamental lemma on NFAs.

Lemma 3. *Let* A *be a* NFA. *Then any word* u *over the alphabet of* A *satisfies:*

$$u^{-1}[\![A]\!] = \cup_{q \in Q_u(A)} [\![A]\!]_q$$

.

Proof. This proof work by recursion on the size of u. For ε, we have $\varepsilon^{-1}[\![A]\!] = [\![A]\!]$, which is $\cup_{q \in Q_I} [\![A]\!]_q$. For starting the induction, we note that $Q_\varepsilon(A) = Q_I$, so that $\varepsilon^{-1}[\![A]\!] = \cup_{q \in Q_\varepsilon(A)} [\![A]\!]_q$. For the induction step, let uf be a word on Σ. We want to find words of $uf^{-1}[\![A]\!]$. We note that $v \in uf^{-1}[\![A]\!]$ if and only if $ufv \in [\![A]\!]$, which is to say $fv \in u^{-1}[\![A]\!]$. We have $u^{-1}[\![A]\!] = \cup_{q \in Q_u(A)} [\![A]\!]_q$ by induction hypothesis, hence there exists $q \in Q_u(A)$ such that $fv \in [\![A]\!]_q$. This in turn means that there exists a rule $q \xrightarrow{f} q'$ such that $v \in [\![A]\!]_{q'}$. Hence this state q' is in $Q_{uf}(A)$, which means that $v \in uf^{-1}[\![A]\!]$ if and only if $v \in \cup_{q' \in Q_{uf}(A)} [\![A]\!]_{q'}$; □

3.1.2 Determinism

We remind the notion of determinism on word automata, and the important properties linked to determinism

Definition 4. *We call a word automaton A* deterministic *(and call it* DFA*), if Q_I contains at most one state, and for any letter f of Σ, and any states q of A, there is at most one rule of form (q, f, q').*

It is important to note the following classical result (see for example [Hopcroft and Ullman, 1979]).

Proposition 5. *If L is a regular language, then there exists a* DFA *A such that $L = [\![A]\!]$.*

In other words, any word automaton can be made deterministic while preserving its language.

Note that if most definitions of NFA and DFA consider word automata as a machine that reads a word from left to right, one can also see them as starting from the right and proceeding to the left. This can be done in two way. The first would be to redefine the way the automaton's semantics are defined:

- if $q \in Q_F$, then $\varepsilon \in [\![A]\!]_q$
- if $(q, f, q') \in rul$, and $u \in [\![A]\!]_{q'}$, then $uf \in [\![A]\!]_q$

An other, equivalent way would be to keep the previously defined semantics, but to define determinism from the final states to the initial states, in a *right-to-left* determinism.

Definition 6. *We call a word automaton A* right-to-left deterministic*, if Q_F contains at most one state, and for any letter f of Σ, and any states q of A, there is at most one rule of form (q', f, q).*

Note that just like left-to-right determinism, these restrictions pose no limitation to the expressiveness of word automata.

Proposition 7. *If L is a regular language, then there exists a right-to-left deterministic* NFA *A such that $L = [\![A]\!]$.*

3.1.3 Myhill-Nerode Theorem

By Myhill-Nerode Theorem, we design not only the specific result of [Nerode, 1958], but more generally a result where a semantic characterisation of a formal object allows for the definition of a normal form.

In the case of word automata, the Myhill-Nerode theorem states that for any regular language L there exists a finite number of residuals $u^{-1}L$. This leads to the existence of a unique DFA with a minimal number of states that defines L.

Definition 8. *Let L be a language, and u, u' two words. We say that $u \equiv_L u'$ if and only $u^{-1}L = u'^{-1}L$.*

We can prove this equivalence relation to be of finite index for regular languages.

Proposition 9. *Let L be a regular word language. Then the equivalence relation $\equiv_L$ is of finite index.*

Proof. This is a direct result of Lemma3. Let A be an automaton such that $[\![A]\!] = L$. Since all residuals $u^{-1}L$ are of form $\cup_{q \in Q_u(A)} [\![A]\!]_q$, then there is at most one residual per possible value of $Q_u(A)$. □

Note that in a DFA, all sets $Q_u(A)$ are either a singleton, or empty, further limiting the number of possible residuals: each residual is of form $[\![A]\!]_q$, or $\emptyset$. This essential property allows for the definition of a unique DFA with a minimal number of states for any language described by a DFA.

Proposition 10. *Let L be a tree language described by a* DFA*. Then there exists a unique minimal* DFA *A such that $[\![A]\!] = L$.*

Proof. Since in a DFA, all residuals are either $\emptyset$ or of form $[\![A]\!]_q$, a DFA would necessarily be minimal if it had exactly one state describing each possible residual of L. We can define such a normal form $A = (Q, Q_I, Q_F, rul)$ as follows:

- The set of states Q is the set of all equivalence classes for $\equiv_L$. We note $[u]_L$ the equivalence class of u for $\equiv_L$.
- The unique initial state of Q_I is $[\varepsilon]_L$
- The final states Q_F is the set of all classes $[u]_L$ such that $u \in L$
- The rules of rul go as follows: for all $[u]_L$, for all f such that $uf^{-1}L \neq \emptyset$:

$$[u]_L \xrightarrow{f} [uf]_L$$

One can prove by induction that $[\![A]\!]_{[u]_L} = u^{-1}L$. We consider a word $w = fv$, such that $w \in u^{-1}L$. This means that $v \in uf^{-1}L$. We suppose by induction that $v \in [\![A]\!]_{[uf]_L}$. Given the existence of the rule $[u]_L \xrightarrow{f} [uf]_L$, this is equivalent to $w \in [\![A]\!]_{[u]_L}$.

Hence $[\![A]\!]_{[u]_L} = u^{-1}L$, and more notably, $[\![A]\!] = [\![A]\!]_{[\varepsilon]_L} = L$. □

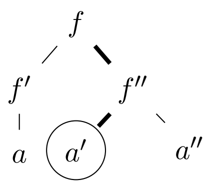


Figure 3.1: The F-path $u = f2f''1$ and the node that it addresses. The F-npath ua' specifies the label a' of this node in addition.

3.2 Trees

In this thesis we will consider ranked trees on a finite signature.

A *ranked alphabet* is an alphabet F together with a total function $rank_F : F \to \mathbb{N}$. If $rank_F(f) = k$, we say that f is of *rank* k, which means it expects k children exactly. For $k \geq 0$ we denote by $F^{(k)}$ the set $\{f \in F \mid rank_F(f) = k\}$. We often write $f^{(k)}$ to indicate that f is of rank k.

Let F be a ranked alphabet. The set of (ordered, finite) trees over F is the set of ground terms over F and denoted by $\mathcal{T}_F$. This is the least set such that for any $f \in F^{(k)}$, $k \geq 0$, and $s_1, \ldots, s_k \in \mathcal{T}_F$, the term $f(s_1, \ldots, s_k)$ belongs to $\mathcal{T}_F$. For a one-node tree $f()$, where f is a constant, we simply write f. For a finite set X disjoint from F we define $\mathcal{T}_F(X)$ as $\mathcal{T}_{F \cup X}$ where any $x \in X$ has rank 0. We define an F-path to be a word with alphabet $F \cup \mathbb{N}$ in $\{fi \mid f \in F^{(k)},\ 1 \leq i \leq k\}^*$. Note that constants of F cannot appear in F-paths. We will always identify nodes of a tree $s \in \mathcal{T}_F$ with the unique F-path that leads to them:

- The root of s is reached by the empty word ε.
- If a node of s is labeled by $f \in F^{(k)}$ and reached by the F-path u, then its i^{th} child is reached by the F-path ufi, where $1 \leq i \leq k$.

An example for a node of a tree s and its corresponding F-path u is given in Fig. 3.1. In this case, we say that u belongs to s and write:

$$s \models u$$

The set of all paths that belong to s is denoted by $paths(s)$. If $s \models u$, then we denote the label of the node addressed by u with $s[u]$ and the *subtree of s at u* by $u^{-1}s$. An example for a subtree is given in Fig. 3.2. We can generalize this notation to tree languages by defining the residual language $u^{-1}L$ as $\{u^{-1}s \mid s \in L, s \models u\}$.

It should be noticed that an F-path does not state anything about the label of the node to which it leads. We introduce node paths for this purpose as follows. An *F-node-path*, or *F-npath* is a word uf starting with an F-path u and ending with a symbol $f \in F$ of arbitrary arity. Note that uf points to a leaf in some tree if and only if f is a constant. We say that $s \models uf$ if $s \models u$ and $s[u] = f$.

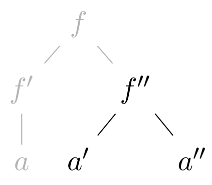


Figure 3.2: The tree s and its subtree at path $f2$, denoted by $(f2)^{-1}s$.

We define two types of substitution on trees. For $s_1, ..., s_n$ such that no s_i is the subtree of another s_j, we denote by $\tau = [s_1/t_1, \ldots, s_n/t_n]$ the finite transformation that maps input trees $s_i \in \mathcal{T}_F$ to output trees $t_i \in \mathcal{T}_F$ for all $1 \le i \le n$. For any $t \in \mathcal{T}_F$, we write $t\tau \in \mathcal{T}_F$ for the tree obtained from t by substituting subtrees s_i by t_i. For $u_1, ..., u_n$ such that no u_i is the prefix of another u_j, we denote by $\alpha = [u_1/s_1, \ldots, u_n/s_n]$ the substitution that maps F-paths u_i to trees s_i for all $1 \le i \le n$. For any tree $s \in \mathcal{T}_F$, we write $s\alpha \in \mathcal{T}_F$ for the tree obtained from s by replacing the subtrees at paths u_i by the trees s_i.

Definition 11. *A tree language $L \subseteq \mathcal{T}_F$ is called* path-closed *if, for any two trees $s, s' \in L$ with $s \models u$ and $s' \models u$, it holds that $s[u/u^{-1}s'] \in L$.*

In other words, for any two trees of a path-closed language, one can exchange their subtrees at the same path and remain in the language. For instance, the language $\{f(a,a), f(a',a'), f(a,a'), f(a',a)\}$ is path-closed, while the language $\{f(a,a), f(a',a')\}$ is not.

3.2.1 Tree Automata

We recall the notion of top-down tree automata, their notion of determinism and the relationship with path-closed languages.

Definition 12. *A* (nondeterministic) tree automaton *is a triple $A = (F, Q, Q', rul)$ composed of*
- *a finite ranked signature F,*
- *a finite set Q of states,*
- *a subset $Q' \subseteq Q$ of root-labeling states, and*
- *a binary relation $rul \subseteq \cup_{k \ge 0}(Q \times F^{(k)} \times Q^)k of transitions.$*

We view tree automata as sets of rules for labeling all nodes of a trees by some state in a nondeterministic manner. A labeling is accepted if the root node is labeled by some state in Q' and if the labels of all nodes are licenced by some rule in rul.

More formally, for any states $q \in Q$ we define membership to the language $[\![A]\!]_q$ accepted by A when started at q by induction on the structure on trees, such that for all $k \geq 0$, $f \in F^{(k)}$ and $s_1, \dots, s_k \in \mathcal{T}_F$:

$$f(s_1, \dots, s_k) \in [\![A]\!]_q \Leftrightarrow_{\text{def}} (q, f, q_1, \dots, q_k) \in \mathit{rul} \text{ and } s_i \in [\![A]\!]_{q_i} \text{ for all } 0 \leq i \leq k.$$

We define the language that A accepts by $[\![A]\!] = \cup_{q \in Q'} [\![A]\!]_q$. A tree language L is called *regular* if $L = [\![A]\!]$ for some tree automaton A.

Top-down versus Bottom-up Tree Automata. In the literature [Comon et al., 2007], tree automata are often viewed as machines that rewrite trees to a state, either in a top-down or in a bottom-up manner. Our notion of tree automaton does not impose any rewrite direction, so it captures both bottom-up and top-down nondeterminsitic tree automata (which are well-known to be equivalent when nondeterministic).

In a top-down tree automaton the set of root states is usually called the set initial states, while in a bottom-up tree automaton it is called the set of final states. We will write the rules of rul in two different ways depending of whether we want to emphasise the top-down or the bottom-up view. A rule $(q, f, q_1, \dots, q_k) \in \mathit{rul}$ in a top-down perspective is denoted by:

$$q \xrightarrow{f} (q_1, \dots, q_k)$$

In bottom up perspective the same rule is denoted by:

$$f(q_1, \dots, q_k) \to q$$

Top-down Reachability. Since we aim to study top-down tree transducers, we will now introduce some terminology to reason on the different labellings a node can get in a tree from a tree automaton, in a top-down manner: for any tree automaton A and path u for the signature of A, we define the set of states *reached (top-down) by u in A* as the set $Q_u(A)$ that A may assign to path u as follows by induction on the length of u:

$$\begin{array}{rcl} Q_\varepsilon(A) & = & Q', \qquad \text{(the root states of } A\text{)} \\ Q_{ufi}(A) & = & \{q_i \in Q \mid q \xrightarrow{f} (q_1, \dots, q_k),\ q \in Q_u(A)\}. \end{array}$$

Definition 13. *A tree automaton $A = (F, Q, Q', \mathit{rul})$ is* trimmed *if for all $q \in Q$:*

bottom-up reachability: $[\![A]\!]_q \neq \emptyset$*, and*

top-down reachability: *there exists a tree $s \in [\![A]\!]$ and a path $u \in \mathit{paths}(s)$ such that $q \in Q_u(A)$.*

Every tree automaton can be made trimmed in linear time by removing all useless states, i.e., those that cannot be reached either top-down or bottom-up. Therefore, we can assume that all tree automata are trimmed whenever this will be needed in what follows. For any trimmed tree automaton A and path u, note that $Q_u(A) \neq \emptyset$ if and only if $u \in paths(\llbracket A \rrbracket)$.

We next state a simple but fundamental lemma on trimmed tree automata.

Lemma 14. *Let A be a trimmed tree automaton. Then any path u over the signature of A satisfies:*

$$u^{-1}\llbracket A \rrbracket = \cup_{q \in Q_u(A)} \llbracket A \rrbracket_q$$

.

Proof. This proof work by recursion on the size of u. For ε, we have $\varepsilon^{-1}\llbracket A \rrbracket = \llbracket A \rrbracket$, which is $\cup_{q \in Q'} \llbracket A \rrbracket_q$. For starting the induction, we note that $Q_\varepsilon(A) = Q'$, so that $\varepsilon^{-1}\llbracket A \rrbracket = \cup_{q \in Q_\varepsilon(A)} \llbracket A \rrbracket_q$. For the induction step, let ufi be a path in F. We have $u^{-1}\llbracket A \rrbracket = \cup_{q \in Q_u(A)} \llbracket A \rrbracket_q$ by induction hypothesis. From there, we have to search for all fi-subtrees. If $t = f(t_1, ..., t_n) \in u^{-1}\llbracket A \rrbracket$, then there exists $q \in Q_u(A)$ such that $t \in \llbracket A \rrbracket_q$. By definition, this means that there is a rule $(q, f, q_1 \cdots q_n) \in rul$, such that $t_1 \in \llbracket A \rrbracket_{q_1}, ..., t_n \in \llbracket A \rrbracket_{q_n}$. This means that notably, all trees of $ufi^{-1}\llbracket A \rrbracket$ are in some $\llbracket A \rrbracket_{q_i}$ such that $q_i \in Q_{ufi}(A)$.

Conversely, if some t_i is in $\llbracket A \rrbracket_{q_i}$ such that $q_i \in Q_{ufi}(A)$, then there is a state $q \in Q_u(A)$ and a rule $(q, f, q_1 \cdots q_n) \in rul$. Since A is trimmed, all $\llbracket A \rrbracket_{q_j}$ are nonempty. This means that we can choose $(t_1, ..., t_{i-1}, t_{i+1}, ..., t_n)$ such that for all j between 1 and n, $t_j \in \llbracket A \rrbracket_{q_j}$. This means that $t = f(t_1, ..., t_n) \in \llbracket A \rrbracket_q$, and thus by induction $t \in u^{-1}\llbracket A \rrbracket$. Finally, if $t \in u^{-1}\llbracket A \rrbracket$, then $t_i \in ufi^{-1}\llbracket A \rrbracket$. □

3.2.2 Determinism

Two non-equivalent kinds of determinisms exist for tree automata: top-down determinism and bottom-up determinism.

Definition 15. *We call a tree automaton A* bottom-up deterministic*, if for any letter f of rank k, and any states $q_1, ..., q_k$ of A, there is at most one rule of form $f(q_1, ..., q_k) \to q$.*

It is important to note the following classical result (see for example [Comon et al., 2007]).

Proposition 16. *If L is a regular language, then there exists a bottom-up deterministic tree automaton A such that $L = \llbracket A \rrbracket$.*

In other words, any tree automaton can be made bottom-up deterministic while preserving its language. The proof is by the usual subset construction, similar to the determinization of NFAs.

Definition 17. *We call a tree automaton A* top-down deterministic *or equivalently a* DTTA*, if its set of initial states Q' contains at most one element and if for an input letter f of rank k, and q a state of Q, there is only one rule of form $q \xrightarrow{f} (q_1, ..., q_k)$.*

Note that top-down determinism translates the fact that for any given tree, the automaton always has at most one choice for labelling the nodes of a tree when processing it from the root to the leaves. To link it to our previous definitions, this means that for every path u the set $Q_u(A)$ is either a singleton, or $\emptyset$.

It is well-known that not all regular languages can be recognized by top-down deterministic tree automata. This property essentially translates the fact that which subtree can be chosen at a path u of a tree of L depends only of the path u but not on the context around u. One can substitute a valid subtree under a path u with any other valid subtree under u regardless of what is elsewhere in the tree. This property is not respected in all regular languages. it is, however, true for all languages accepted by a DTTA.

However, the class of languages accepted by a DTTA are path-closed.

Lemma 18. *Any language accepted by some* DTTA *is* path-closed.

Proof. Let A be a trimmed DTTA, $u \in \mathit{paths}(\llbracket A \rrbracket)$, and $s \in \llbracket A \rrbracket$ such that $s \models u$. Since A is top-down deterministic, then for every u the set of u-accessible states Q_u is either empty or a singleton. As seen in Lemma 14, we can replace $u^{-1}s$ by any other tree t in $\llbracket A \rrbracket_q$, and have $s[u/t] \in \llbracket A \rrbracket$. Since $u^{-1}\llbracket A \rrbracket \subseteq \llbracket A \rrbracket_q$, we obtain that $\llbracket A \rrbracket$ is path-closed. □

The converse of Lemma 18 does not hold. But it is true that a language is recognizable by some DTTA if and only if it is regular and path closed.

3.2.3 Myhill-Nerode Theorem

A Myhill-Nerode theorem exists for both top-down deterministic and deterministic bottom-up automata. They allow the definition of a minimal normal form.

Top-Down Deterministic Tree Automata The Myhill-Nerode theorem on top-down deterministic tree automata works using the notions of top-down residuals ($u^{-1}L$) we defined above. We define the equivalence relation $\equiv_L$ as follows:

Definition 19. *Let L be a tree language, and u, u' two paths. We say that $u \equiv_L u'$ if and only $u^{-1}L = u'^{-1}L$.*

We can prove this equivalence relation to be of finite index.

Proposition 20. *Let L be a regular tree language. Then the equivalence relation $\equiv_L$ is of finite index.*

Proof. This is a direct result of Lemma 14. Let A be an automaton such that $\llbracket A \rrbracket = L$. Since all residuals $u^{-1}L$ are of form $\cup_{q \in Q_u(A)} \llbracket A \rrbracket_q$, then there is at most one residual per possible value of $Q_u(A)$. □

Note that in a DTTA, all sets $Q_u(A)$ are either a singleton, or empty, further limiting the number of possible residuals: each residual is of form $[\![A]\!]_q$, or $\emptyset$. This essential property allows for the definition of a unique DTTA with a minimal number of states for any language described by a DTTA.

Proposition 21. *Let L be a tree language described by a* DTTA*. Then there exists a unique minimal* DTTA *A such that $[\![A]\!] = L$.*

Proof. Since in a DTTA, all residuals are either $\emptyset$ or of form $[\![A]\!]_q$, a DTTA would necessarily be minimal if it had exactly one state describing each possible residual of L. We can define such a normal form $A = (Q, Q', rul)$ as follows:

- The set of states Q is the set of all equivalence classes for $\equiv_L$. We note $[u]_L$ the equivalence class of u for $\equiv_L$.
- The unique initial state of Q' is $[\varepsilon]_L$
- The rules of rul go as follows: for all $[u]_L$, for all f such that $uf \in paths(L)$:

$$[u]_L \xrightarrow{f} ([uf1]_L, ..., [ufn]_L)$$

One can prove by induction that $[\![A]\!]_{[u]_L} = u^{-1}L$. We consider a tree $t = f(t_1, ..., t_n)$, such that $t \in u^{-1}L$. Since L is path-closed, this is equivalent to say that for all i from 1 to n, $t_i \in ufi^{-1}L$. By induction, this is equivalent to $t_i \in [\![A]\!]_{[ufi]_L}$. By the definition of A, this is equivalent to $t \in [\![A]\!]_{[u]_L}$.

Hence $[\![A]\!]_{[u]_L} = u^{-1}L$, and more notably, $[\![A]\!] = [\![A]\!]_{[\varepsilon]_L} = L$. □

Bottom-Up Deterministic Tree Automata The Myhill-Nerode theorem on bottom-up deterministic tree automata works using a different equivalence class: two trees t, t' are said to be equivalent for a language L if for any tree $s \in L$ such that $u^{-1}s = t$, then $s[u/t'] \in L$ and reciprocally, for any tree $s \in L$ such that $u^{-1}s = t'$, then $s[u/t] \in L$.

The full proof of the existence of a minimal normal form for bottom-up deterministic tree automata is available in the literature (e.g. [Comon et al., 2007]), and relies on the finite index of the relation described above, and the fact that each state of a bottom-up deterministic tree automaton describes exactly one such equivalent class.

Proposition 22. *Let L be a tree language described by a bottom-up deterministic tree automaton. Then there exists a unique minimal bottom-up deterministic tree automaton A such that $[\![A]\!] = L$.*

3.3 Automata Learning

The class of regular languages represented by DFAs is learnable in Gold's model with polynomial resources [Oncina and Garcia, 1992] by an algorithm called *Regular Positive and Negative Inference* (RPNI). In this section we recall Gold's model and the RPNI algorithm for regular word languages from positively or negatively annotated examples.

The RPNI algorithm has been lifted to various other kinds deterministic machines. We only present the case DFAs since the main ideas can be seen there. There are RPNI style algorithms for regular tree languages defined by bottom-up deterministic tree automata [Oncina and García, 1993, Carme et al., 2007, Niehren et al., 2013], top-down regular tree languages defined by top-down deterministic tree automata, and determinstiic (subsequential) transducers on words [Oncina et al., 1993]. The latter variant of RPNI is called OSTIA. Finally, there is a RPNI style learning algorithm for tree transformations defined by DTOPs [Lemay et al., 2010] that we will revisit in detail later on in this thesis.

3.3.1 Learning Model

We fix a finite alphabet Σ. A *sample* $S = (S_+, S_-)$ is the pair formed by S_+ a set of positive examples, and S_- a set of negative examples A sample for a regular language L is a sample $S = (S_+, S_-)$ such that $S_+ \subseteq L$ and $S_- \cap L = \emptyset$.

We now give a definition of learning function and characteristic sample. The idea behind those is that the learning function *learn* is supposed to send a sample for L to the minimal normal DFA that describes L if S contains enough information. This notion of enough information is made formal using the notion of characteristic sample: if S is a characteristic sample for L, then $learn(S)$ will indeed find the minimal normal DFA that describes L.

Definition 23. *We say that the class of* DFA*s is learnable if there are:*

- *an algorithm learn defining a partial function that maps samples to* DFA*s in normal form, and*
- *a function char that maps* DFA*s A in normal form to samples of transformation $[\![A]\!]$.*

We require for any DFA *A and any sample S for $[\![A]\!]$ containing $char(A)$ that $learn(S) = A$.*

There are several parameters to consider when describing the complexity of learning algorithms:

- Sample complexity describes the number of examples in $char(A)$ as a function of the size of A.
- Time complexity describes the complexity of the learning algorithm *learn* as a function of the size of its input sample.

We say a class is learnable *with polynomial ressources* if the the number of examples in $char(A)$ is polynomial as a function of the size of A, and the learning algorithm *learn* is in polynomial time as a function of the size of its input sample.

The Rpni algorithm is indeed such an algorithm. We will present the characteristic sample and learning algorithm in this section, proving the following result:

Theorem 24. *The class of* Dfa*s is learnable with polynomial ressources.*

3.3.2 Characteristic Samples

The purpose a characteristic sample to provide enough information to describe a Dfa A in normal form. Since the minimal normal form of Proposition 10 is based on the finitely many equivalence classes of $\equiv_{[\![A]\!]}$, we want to identify all these classes and their relationship. This is done in practice by finding one representant in each equivalence class.

The first question is by which word u to represent each class of $\equiv_{[\![A]\!]}$. The idea is to choose the least word u' that defines the same residual language $u'^{-1}[\![A]\!] = u^{-1}[\![A]\!]$ with respect to the following total order. If u, v two words of Σ, we define $u < u'$ if and only if $|u| < |u'|$, or $|u| = |u'|$ and $u <_{lex} u'$.

To properly define the notion of a sample S containing enough information to learn a Dfa A, we will establish what prefixes of words of L are of relevance, and what S should teach on them. Our first move is to define minimal words and their boundary. Minimal words are the least representant of the equivalence classes of $\equiv_{[\![A]\!]}$, and their boundary are all their direct successors.

Definition 25. *For any regular language L and u a word such that $u^{-1}L \neq \emptyset$, we define $minw_L(u) = min\{u' \mid u' \equiv_L u\}$. By extension, we define the set of minimal words of L as:*

$$minw(L) = \{minw_L(u) \mid u^{-1}L \neq \emptyset\}$$

Note that for any Dfa A the set of residuals of $[\![A]\!]$ is finite by Proposition 9. This means the set $minw(L)$ for a regular language L is finite too.

To find all minimal words, our algorithm will explore words starting at ε, and extend its exploration by adding one letter at the end. If any new minimal word is found, it will then continue the exploration by adding one letter at the end of this new minimal word, etc. Hence, the words that will be explored do not only contain those in $minw(L)$, but also the words obtained by adding one letter to them.

Definition 26. *For any transformation L we define the boundary of the minimal words of L as:*

$$minw^+(L) = \{\varepsilon\} \cup \{ua \mid a \in \Sigma,\, u \in minw(L),\, ua^{-1}L \neq \emptyset\}$$

The following lemma shows for regular languages L, all elements of $minw(\tau)$ are in fact either ε or of form ua such that $u \in minw(L)$:

Lemma 27. *Let A be a* DFA*, and $[\![A]\!] = L$. Then $minw(L) \subseteq minw^+(L)$*

Proof. Let u be a minimal word of L. If $u = \varepsilon$ then $u \in minw^+(L)$. If $u = u'a$, then we show that $u' \in minw(L)$. If $u' \notin minw(L)$, then we call $v = minw_L(u')$. This means that $v < u'$ and $v^{-1}L = u'^{-1}L$. By adding the letter a to both we get that $va < u'a$ and $va^{-1}L = u'a^{-1}L$. This goes against the fact that $u'a = u$ is a minimal word. Hence $u' \in minw(L)$, which means that $u = u'a \in minw^+(L)$. □

The set $minw(L)$ is designed to have one unique representative for each class of $\equiv_L$. In this sense, it can be seen as representing the states of the unique minimal DFA describing L. As a matter of fact, we will define $repdfa(L)$ the state renaming of the normal form described in Proposition 10 where each state $[u]_L$ is represented by the unique minimal word $u' = minw_L(u)$.

Definition 28. *Let A a* DFA*, and $L = [\![A]\!]$. We define $repdfa(L)$ the minimal normal representative of A as the* DFA *$A = (Q, Q_I, Q_F, rul)$ where:*
- *$Q = minw(A)$*
- *$Q_I = \{\varepsilon\}$*
- *$Q_F = \{u \in Q \mid \varepsilon u^{-1}L\}$*
- *For $u \in minw(L)$, a such that $ua^{-1}L \neq \emptyset$,*

$$u \xrightarrow{a} ua$$

This $repdfa(L)$ is the actual target of our algorithm. Each minimal word has a state in $repdfa(L)$, each equivalence $u \equiv_L u'a$ creates a transition in $repdfa(L)$.

We now consider our sample S, and establish what kind of information it needs to contain for our learning algorithm to be able to retro-engineer $repdfa(L)$. To this end we will notably have to be able to differentiate words of $minw^+(L)$ that are not equivalent. In order to show that $u \not\equiv_L u'$ by a sample S for L, we will require that u and u' are in contradiction with respect to S in the following sense:

Definition 29. *Given a sample $S = (S_+, S_-)$ for a transformation L, we say that two words u and u' are in contradiction with respect to S and write $u \nparallel_S u'$ if there exists a word $v \in \Sigma^*$ such that $uv \in S_+$ and $u'v \in S_-$, or $u'v \in S_+$ and $uv \in S_-$.*

In this case, S contains a counter example for $u \not\equiv_\tau u'$. We now use this definition to formalize what it means for a sample S to be characteristic for L.

Definition 30. *Let A be a* DFA *and $L = [\![A]\!]$. A sample S for L is called* characteristic *if:*

(1) For all $u \in minw^+(L)$, $u^{-1}S_+ \neq \emptyset$.
(2) For all $u \in minw(L)$ such that $\varepsilon \in u^{-1}L$, $u \in S_+$.
(3) For all $u \in minw^+(L)$, and all $u' \in minw(L)$ such that $u \not\equiv_L u'$: $u \not\parallel_S u'$.

Points (1) ensures that S knows which words of form ua such that $u \in minw(L)$ correspond to the empty residual, i.e. are not in $minw^+(L)$. Points (2) ensures that S knows which words of $minw(L)$ correspond final states. Point (3) ensures we are able to tell which words are equivalent to which minimal words.

Note that if a sample S for L is characteristic, then any larger sample for L is characteristic too. It is quite natural that all DFAs A possess a characteristic sample of polynomial size.

Proposition 31. *For any* DFA *A, where $[\![A]\!] = L$, there exists a characteristic sample for L with a number of examples polynomial in the number of equivalence classes in $\equiv_L$.*

Proof. We start by noting that $minw(L)$ has one element per class of $\equiv_L$, and $minw^+(L)$ has one element per class of $\equiv_L$ and letter of Σ. We effectively prove our sample to be polynomial in the size of $minw^+$

For point (1), we only require one positive example per $u \in minw^+(L)$ such that $u^{-1}L \neq \emptyset$, and thus no more than $\#minw^+(L)$ examples.

For point (2), we only require one positive example per $u \in minw(L)$ such that $\varepsilon^{-1}L$, and thus no more than $\#minw^+(L)$ examples.

For point (3), we require one positive example and one negative example for each pair (u, u') such that $u \in minw^+(L)$, and all $u' \in minw(L)$ such that $u \not\equiv_L u'$. Thus we need no more than $2\#minw^+(L)^2$ examples. □

3.3.3 Learning Algorithm RPNI

We describe the algorithm *learn*. The goal is to create a minimal normal DFA, which means creating states with no redundancy, their transitions, and identifying the final states. The idea is to try to fold any new word we find to an existing state. If no equivalent state can be found, we create a new one.

In this algorithm we build a DFA $learn(S) = (Q, Q_I, Q_F, rul)$. For simplicity's sake, our states will be words $u \in \Sigma^*$. Those states will be divided in two disjoint sets: Q_{safe} for pairs that minimally represent an equivalence class, and therefore represent a state in $learn(S)$, and Q_{temp}, for pairs that have not yet been examined by the algorithm, and are still susceptible to be equivalent to an existing pair in Q_{safe}. Rules of rul are only created for states of Q_{safe}, but they can potentially call "unapproved" states of Q_{temp}.

Procedure *integrate-state* describes how, given a DFA A and a sample S, to test if a word $u \in Q_{temp}$ is equivalent to an existing state in Q_{safe} and, if it is not, how to create a new state and its rules. Note that in these rules, $ua\langle x_i\rangle$ can appear for words ua that are not yet confirmed to be original states. These pairs are added to Q_{temp}. The final states are all states $u \in Q_{safe}$ such that $u \in S_+$ From there, the full algorithm goes as described in Figure 3.3.

Our end goal in this part is to prove the correctness of our algorithm, i.e. that if A is a DFA, $L = [\![A]\!]$, S is a characteristic sample for L, then $learn(S)$ is equal to $repdfa(L)$.

To this end, we will show that at each intermediary step of the algorithm, we will learn a "partially unfolded" version of $repdfa(L)$: before calling *integrate-state* on a word u, for all minimal words u' such that $u' < u$, u' should be in Q_{safe}, and ε and all words of form $u'a$ should have appeared in Q_{temp}, but all such word smaller than u should already have been integrated. This leads to a DFA that has some definitive states in Q_{safe}, some unexplored temporary states in Q_{temp}, and some rules of rul leading to a safe state or a temporary state, depending on their lexical order relative to u.

To formalize this notion of partially unfolded rules, we first define what pairs should be replaced, what pairs should still be unexplored, and we define the u-*truncated* version of $repdfa(L)$.

Definition 32. *Let A be a* DFA*, $L = [\![A]\!]$, and u such that $u^{-1}L \neq \emptyset$. For $u' \in minw^+(L)$, we call $minrep^u_L(u')$ the u-truncated representative of u':*

- *$minrep^u_L(u') = minw_L(u)$ if $u' < u$,*
- *$minrep^u_L(u') = u'$ itself if $u' \geqslant u$*

Definition 33. *Let A be a* DFA*, $L = [\![A]\!]$, $repdfa(L) = (Q, Q_I, Q_F, rul)$, and u a word of L. We define the u-truncated form of repdfa(L), $repdfa_u(L) = (Q_{safe(u)} \cup Q_{temp(u)}, Q_I, Q_{F<u}, rul_u)$, where:*

- *The u-truncated safe states $Q_{safe(u)} = \{u' \in Q \mid u' < u\}$*
- *The u-truncated temporary states*
  $Q_{temp(u)} = \left(\{\varepsilon\} \cup \{u'a \mid u' \in Q_{safe(u)},\ u'a^{-1}L \neq \emptyset\}\right) \cap \{u' \mid u' \geqslant u\}$
- *The u-truncated finite states $Q_{F<u} = \{u' \in Q_F \mid u' < u\}$*
- *The u-truncated rules rul_u are defined so that if $u' \in Q_{safe(u)}$, and a a letter such that $u'a^{-1}L \neq \emptyset$, then*

$$u' \xrightarrow{a} minrep^u_L(u'a)$$

Note that the u-truncated form "develops" as u grows, to finally become $repdfa(L)$ itself if $u > max(minw^+(L))$.

Corollary 34. *Let A be a* DFA*, u a word of L such that $u > max(minw^+(L))$. Then $repdfa_u(L) = repdfa(L)$.*

```
// let $\Sigma$ be a finite alphabet

fun $learn(S)$ // $S = (S_+, S_-)$ positive and negative examples
  $Q_{temp}$ := $\{\varepsilon\}$
  $Q_I$ := $\{\varepsilon\}$
  $Q_F$ := $\emptyset$
  $Q_{safe}$ := $\emptyset$
  $rul$ := $\emptyset$
  $A$ = $(Q_{safe} \cup Q_{temp}, Q_I, Q_F, rul)$
  proc $integrate\text{-}state(u)$ = // fusion temporary state $u$ with
      // some safe state if possible or make $u$ safe
      // and create its transitions.
    $Q_{eq}$ = $\{u' \in Q_{safe} \mid$ not $u \nparallel_S u'\}$
  in
    case $Q_{eq}$ // $Q_{eq}$ may contain at most 1 element
    of $\{u'\}$ then replace all occurrences $of\ u\ in\ A$ by $u'$
                      $Q_{temp}$ := $Q_{temp} \backslash \{u\}$
    of $\emptyset$ then
      $Q_{safe}$ := $Q_{safe} \cup \{u\}$
      $Q_{temp}$ := $Q_{temp} \setminus \{u\}$
      if $u \in S_+$ then $Q_F$ := $Q_F \cup \{u\}$
      for $a \in \Sigma$ where $ua^{-1}S_+ \neq \emptyset$ do
        $Q_{temp}$ := $Q_{temp} \cup \{ua\}$
        $rul(u,a)$ := $ua$
      end
    end
  end
in
   while $Q_{temp} \neq \emptyset$ do
     $u$ = $min(Q_{temp})$
   in
     $integrate\text{-}state(u)$
   end
   return $(A)$
end
```

Figure 3.3: Learning algorithm of DFAs.

Proof. $Q_{safe(u)} = minw(L)$, $Q_{temp(u)} = \emptyset$, $Q_{F<u} = Q_F$ hence $repdfa_u(L)$ and $repdfa(L)$ have same states, initial states, and final states. Finally, the only difference between the definition of the rules of $repdfa_u(L)$ and $repdfa(L)$ is that the former uses $minrep^u_L$ and the latter uses $minw_\tau$. By construction, if $u > max(minw^+(L))$, for all pairs $p' \in minw^+(L)$, $minrep^u_L(u') = minw_L(u')$. Hence we have that $repdfa_u(L)$ and $repdfa(L)$ have same rules. □

We prove the correctness of *learn* if the sample S is characteristic. To this end, we show that right before each call to *integrate-state*(u), the automaton A created by *learn* is exactly $repdfa_u(L)$.

Proposition 35. *Let A be a* DFA*, $A = [\![A]\!]$. Then $learn(S) = repdfa(L)$.*

Proof. We prove the following invariant: if $Q_{temp} \neq \emptyset$ and $u = min(Q_{temp})$, then $Q_{safe} = Q_{safe(u)}$, $Q_{temp} = Q_{temp(u)}$, $Q_F = Q_{F<u}$ and $rul = rul_u$.

After the initialization, $Q_{safe} = \emptyset$ and $Q_{temp} = \{\varepsilon\}$, which indeed match $Q_{safe()}\varepsilon$ and $Q_{temp()}\varepsilon$. $Q_F = \emptyset$, which indeed matches $Q_{F<\varepsilon}$. As for rules, none have been created yet, which means $rul = rul_u = \emptyset$.

For the inductive case, we consider u the minimal element of Q_{temp}. By induction, we have $Q_{safe} = Q_{safe(u)}$, $Q_{temp} = Q_{temp(u)}$, $Q_F = Q_{F<u}$ and $rul = rul_u$. We call the new values after *integrate-state*(u) Q'_{safe}, Q'_{temp}, Q'_F and rul'. If $Q'_{temp} \neq \emptyset$, we call u' its minimum. u is added to Q_{safe} if and only if for all $u'' \in Q_{safe}$, $u \not\parallel_S u''$. Since S is characteristic, $Q_{safe} \subseteq minw(L)$ and $u \in minw^+(L)$, this means that for all $u'' \in Q_{safe(u)}$, $u \not\equiv_\tau u''$. Hence, u is added to Q_{safe} if and only if $u \in minw(L)$. For the same reason, states are added to Q_{temp} if and only if $u \in minw(L)$. If S is characteristic, then the new states are words of form ua such that $ua^{-1}S_+ \neq \emptyset$. Since S is characteristic and $u \in minw(L)$, this means that $ua \in minw^+(L)$. In all cases u is removed from Q_{temp}. This means that regardless of weather u was added or not, $Q'_{safe} = minw(L) \cap \{u'' \mid u'' \leqslant u\} = minw(L) \cap \{u'' \mid u'' < u'\}$. This means that $Q'_{safe} = Q_{safe(u')}$. If u is not minimal, $Q'_{temp} = Q_{temp} \cap \{u'' \mid u'' > u\} = Q_{temp} \cap \{u'' \mid u'' \geqslant u'\}$ and thus $Q'_{temp} = Q_{temp(u')}$. If u is minimal, $Q'_{temp} = (Q_{temp} \cup \{ua \mid ua^{-1}L \neq \emptyset\}) \cap \{u'' \mid u'' > u\}$. We properly accepted the successors of the new minimal u, and thus $Q'_{temp} = Q_{temp(u')}$. For Q'_F, u is added if and only u is minimal and $u \in S_+$. Since S is characteristic, this is equivalent to u being a final state of $repdfa(L)$. Hence, $Q'_F = \{u'' \in Q_F \mid u'' \leqslant u\}$. This means that $Q'_F = \{u'' \in Q_F \mid u'' < u'\} = Q_{F<u'}$. For the rules, if u is not minimal, there is no new rules, and the occurences of u are replaced by an element of Q_{safe} equivalent to u. Since S is characteristic and $Q_{safe} = Q_{safe(u)}$, this means that the occurences of u are replaced by $minrep_L(u)$. The only difference between rul_u and $rul_{u'}$ is that $minrep^u_L(u') = u'$, while $minrep^{u'}_L(u') = minrep_L(u')$. Hence, $rul = rul_{u'}$. If u is minimal, there is no change between $minrep^u_L$ and $minrep^{u'}_L$ $rul(u,a) = ua$ for $ua^{-1}S_+ \neq \emptyset$. Since S is characteristic, that is to say $ua^{-1}L \neq \emptyset$. The only difference between rul_u

and $rul_{u'}$ is that since u is a new pair in $Q_{safe(u')}$, we need to add the rules $rul(u,a) = ua$ for $ua^{-1}L \neq \emptyset$. Hence, $rul = rul_{u'}$.

It remains to show that the last step that eventually empties Q_{temp} leads to $repdfa(L)$. In all other cases, we consider u the last word to be integrated by $integrate\text{-}state(u)$. Since it is the last considered pair, we have that $u = max(minw^{+}(L))$. As seen in this proof, after this last $integrate\text{-}state(u)$, we have $learn(S) = repdfa_{u'}(L)$, where u' is the word right after u in lexical order. Thus, Corollary 34 gives us that $learn(S) = repdfa(L)$. □

Chapter 4

A Learning Algorithm for Top-Down Tree Transducers

Abstract. A generalization from string to trees and from languages to transformations is given of the classical result that any regular language can be learned from annotated examples: we show how to learn top-down deterministic tree transducers (DTops) in Gold's model with polynomial resources from samples of input-output examples, while assuming a top-down schema for the input domain. Until now, similar results were known only for string transducers, and simple relabeling tree transducers. Learning of DTops is more involved because a DTop can copy, delete, and permute its input subtrees. Thus, complex dependencies of labeled input to output paths need to be maintained by the algorithm.

First, a Myhill-Nerode theorem is presented for DTops with top-down domain inspection, which characterizes the unique normal forms of such machines from [Engelfriet et al., 2009] in a purely semantical manner. This theorem is then used to construct a learning algorithm for DTops, for a given top-down domain inspection. Finally, it is shown how our result can be applied to learn Xml transformations reminicent to Xslt programs, under the assumption that the schema for the domain of input trees is a Dtd or an Xml Schema. For this, a new schema-based encoding of unranked trees by ranked ones is presented. Over such encodings, DTops can realize many practically interesting Xml transformations which cannot be realized on first-child/next-sibling encodings.

This chapter extends a paper presented at PODS'2010 [Lemay et al., 2010] before the beginning of the present thesis. Compared to there, the notion of compatible DTops is generalized so that it extends to arbitrary domain inspections. It is then shown that any DTop with regular domain inspection can be made compatible and earliest (Proposition 69). The more restricted case with top-down inspection was treated already in [Engelfriet et al., 2009], but based on the notion of uniformness, which is a syntactic counterpart to our notion of compatibility. Our generalization prepares a normal form and a learning algorithm for DTops with regular inspection presented in Chapter 5.

Contents

4.1 Introduction 48

4.2 Illustration of Ideas 52

4.2.1 Myhill-Nerode Theorem for DTops 52

4.2.2 Example Transformation 53

4.3 Top-Down Tree Transducers 55
4.3.1 Top-Down Tree Transducers 55
4.3.2 Top-Down Domains 57
4.3.3 Domain Inspection 58
4.4 Syntactic Equivalence 60
4.4.1 Syntactic Alignment 60
4.4.2 Trimmed Transducers 62
4.4.3 Origins of Output Constructors 63
4.4.4 Syntactic Equivalence 64
4.5 Compatible Transducers 65
4.6 Earliest Transducers 68
4.7 Semantic Equivalence 73
4.7.1 Semantic Alignments 73
4.7.2 Semantic Equivalence 75
4.7.3 Semantic Successors 77
4.8 Unique Normal Forms 80
4.9 Learning from Examples 84
4.9.1 Learning Model 84
4.9.2 Characteristic Samples 85
4.9.3 Learning Algorithm 90
4.10 Remaining Proofs 95
4.10.1 Trimming a DTopI 95
4.10.2 Origins of Output Constructors 96
4.11 Conclusions 99

4.1 Introduction

XML is a widely used format for exchanging semi-structured data between programs in various languages, for storing semi-structured data in databases, and for writing all kinds of documents. Specific XML schemas where proposed by various communities for representing their semi-structured data: XBRL for business data, SPL for pharmaceutical products, or SBML for reaction networks in systems biology, for example. The XML format is also omnipresent in Web pages, given that HTML5 is the language supported by all today's Web navigators. Indeed, HTML5 documents are XML documents satisfying yet another XML schema (in contrast to the original HTML).

Document processing is an application area of the XML technology, which includes the creation of web pages and of software documentations. Such documents are usually developped on a higher level of abstraction, supported by tools such as context management systems (Wiki, Plone, Wordpress, etc), or by some other document generator (Docbook). A prime idea there is to separate the document's content and its layout. The pure content can again be represented in XML, but with a different schema that reflects the semantics of the content. Only for displaying the content of a document by a Web browser, the content gets enriched by some layout information. This requires to convert an XML document satisfying the content's schema into another XML document satisfying the schema of HTML5.

As illustrated at the task of Web page publication, the main task to be solved in the context of XML is the conversion of XML documents from one schema into another. Similarly, XML transformations need to be defined for composing two programs in different languages, that support XML formats for input and output, or when exporting the result of a query to an XML database. XML transformations can be defined by programing in general purpose programming languages, or by using dedicated XML transformation languages such as the the W3C standards XSLT 3.0. Here, the acronym XSLT stands for XSL Transformations, where XSL means EXtensible Stylesheet Language.

A major drawback of the programming language approach is that programming is accessible only to programmers, but not by arbitrary Web users. However, imagine a system that is able to automatically infer an XSLT program from a given set of examples. It would free the web programmer from the tedious task of XSLT programming, or the usage of fixed transformations as provided by content management systems. In this chapter we present a learning algorithm that can serve as foundation for building such systems. Under the assumption that the schema of the domain of the target tree transformation is given, our algorithm works in a Gold style model in polynomial time and with polynomially many examples [Gold, 1978]: it takes as an input a finite set of pairs of input and output trees of a target transformation, and, if the input is rich enough, can infer a representation of the target.

In order to make this possible, we need to fix a suitable class of machines for defining tree transformations that we want to learn. Besides *(1) decidable equivalence* in the class of machines to be learned we require that the number of examples needed to to learn an machine of size n is *(2) bounded polynomially in n* (polynomial data) and that the learning algorithm *(3) infers the machine in time polynomial in n*. All three points are closely related to efficient normalization of the machines, which is usually based on a Myhill-Nerode like theorem as for deterministic finite automata (DFA). Given a word language L, two strings x and y are called "L-equivalent" if there is no z such that exactly one of xz and yz is in L. The Myhill-Nerode theorem says that the L-equivalence is of finite index if and only if L is regular. Moreover, there exist unique minimal DFA recognizing L, whose states are exactly the L-equivalence classes.

Since XSLT programs are Turing complete [Kepser, 2004, Onder and Bayram, 2006], polynomial exact learning with (1), (2), and (3) can only be expected for subclasses. The navigational core of XSLT can conveniently be modeled by tree transducers [Bex et al., 2002, Janssen et al., 2007, Maneth et al., 2005, Maneth and Neven, 1999]. For example, macro tree transducers (MTTs) [Engelfriet and Vogler, 1985, Maneth et al., 2005] are a very expressive class of top-down tree transducers than can model a substantial amount of XSLT tree transformations. This expressiveness, however, comes at a huge computational cost: the equivalence problem is a long-standing open question on this class. One of the only progress made in that domain is decidability of a fragment of this class [Engelfriet and Maneth, 2003, Engelfriet and Maneth, 2005], through a non-elementary translation into MSO-definable transformations, and the decidability of (deterministic) top-down tree-to-word transducers with concatenantion [Seidl et al., 2015] which can be simulated by MTTs.

The most well-known and largely-studied class of tree transducer for which equivalence is decidable [Ésik, 1980] is that of deterministic top-down tree transducers (DTOPs). Engelfriet, Maneth and Seidl [Engelfriet et al., 2009] showed recently that DTOPs have a unique normal form, under the condition that the domain of input trees is checked externally. For this, they consider DTOP*s with top-down inspection*, where the schema of input trees is checked by an external deterministic top-down tree automaton. The unique normal form a DTOPs with top-down inspection can be computed by a sequence of syntactic normalization steps: first, these machines are made earliest in the output production, second, they are made uniform with respect to the top-down inspection, and third, their number of states is minimized.

Results on symbolic learning for transducers are few and far between. The OSTIA algorithm [Oncina et al., 1993] allows for polynomial learning of subsequential string transducers in their earliest minimal normal form [Choffrut, 2003]. As for trees, if bottom-up tree automata [Oncina and Garcia, 1992, Carme et al., 2007] in their minimal deterministic normal form can be learned in polynomial time, very few results exists for tree transformation learning. However, these pre-existing results can be seen to have enough of a pattern to hope an extension to the DTOP class: once a Myhill-Nerode theorem is found, some RPNI-like algorithm allows for symbolic learning from a finite sample. The main problem with adapting the normalization procedure of [Engelfriet et al., 2009] in a Myhill-Nerode theorem is its syntactic nature. Therefore it was unclear whether such a normal form can be inferred from a finite sample of input-ouput examples, by generalizing the RPNI algorithm. In the present chapter, we show that this is indeed the case. For this, the following contributions are given:

1. Present a Myhill-Nerode theorem for DTOPs with top-down inspection, which recasts their unique normal forms in a purely semantical terms.

2. For a fixed top-down inspection L, provide a learning algorithm for DTOPs with inspection by L that satisfies Gold's learning model with polynomial resources.

3. Show how to apply our results to the learning of XML transformations. In this case, the top-down inspection is defined by either a DTD or an XML SCHEMA.

Our results are a breakthrough in transducer learning: previous work only considered non-copying and non-swapping transducers (such as word transducers, sequential tree-to-word transducers, or relabeling tree transducers). In contrast, DTOPs have the power to delete, exchange, and copy their input subtrees. Note that many practical XSLT programs make use of deletion and copying of subtrees.

Compared to previous work of Engelfriet, Maneth, and Seidl [Engelfriet et al., 2009] we consider in this article DTOPs with general domain inspection, rather than top-down inspection by top-down deterministic tree automata. This enables us to introduce the notion of *compatible* DTOP*s with inspection*, a semantic generalization of *uniform* DTOP*s with top-down inspection*. We then show that any DTOP with regular inspection can be made compatible and earliest (while generalizing the previous analogous result for top-down inspection). The normal form we present for DTOPs here remains restricted to top-down inspection.

Compared to [Lemay et al., 2010] onto which this article extends, we present a generalized and fully independent proof, rather than a reduction to the results of [Engelfriet et al., 2009]. At the same time, we generalize many of the intermediate results from top-down to regular inspection. While the additional generality is of interest on its own right and lies the foundation for the results of the nect chapter, this approach also yields shorter and simpler proofs, basically due to the usage of the notion of compatibility instead of uniformity.

Related Work

In the context of XML, there has been some work on learning node selection queries [Carme et al., 2007, Niehren et al., 2013, Bonifati et al., 2015, Bonifati et al., 2016, Staworko and Wieczorek, 2015], but only very few on learning tree transformations. The only work on learning XSLT programs that we are aware of is the "XSLT Inference Tool" (part of the Word 2003 SDK by Microsoft). It can infer very restricted types of XSLT programs from a only a single example input and output document pair. The most related work is XLearner [Morishima et al., 2004]. XLearner is a practical system that infers XQuery programs. It uses Angluin's algorithm [Angluin, 1987] in order to infer path DFA's, from which it then constructsXPath expressions. For typical XQueries, the system needs a large number of user interactions (in the hundreds). It seems that the classes of XQuery that are learned by XLearner are incomparable to the class of programs the we infer. As mentioned in [Morishima et al., 2004], there exists interesting work on inferring schema mappings, e.g., LSD [Doan et al., 2001] and Clio [Popa et al., 2002]. It will be interesting to see if an implementation of our results can be useful for automatic inference of XML schema mappings, and if so, how it compares to the such existing systems. There is a large amount of work on learning of DTDs and Schemas, see, e.g., [Bex et al., 2008] and the references given there. It is easily possible to combine any DTD inference algorithm with our work, by simply first inferring input (and output) DTDs, and then executing our algorithm to infer a transformation.

For finite-state transducers, algorithms exist for learning of subsequential string transducers [Oncina et al., 1993]. They are based on minimal earliest transducers, which were formally introduced for strings in [Mohri, 2000], see [Choffrut, 2003] for a survey. A learning algorithm and experimental results for deterministic Mealy machines is presented in [Niese, 2003]. Note that our result, applied to tree translations over monadic trees, also allows to infer minimal string transducers. For tree transducer, the only existing work deals with node selecting queries [Carme et al., 2007], which, in our context can be seen as simple relabelings (that is, DTOPs without copying and permuting of input variables). Previous work on induction of weighted tree transducers compute optimal weights for the rules of a fixed given tree transducer [Graehl et al., 2008].

Outline

In Section 4.2 we start with an illustration of the learning algorithm and its relationship to the Myhill-Nerode theorem that we want to develop. In Section 4.3, we introduce the class of top-down tree transducers with general inspection (DTopI). In Section 4.4 we define the syntactic alignment computed by a DTopI, as a relation that matches paths of input trees to the corresponding output paths as produced by a DTopI. Then, Sections 4.5 and 4.6 will introduce for all DTopI with regular inspection two important normalization steps: compatibility, which ensures that the states of a transducer are as specific as they can be on the input tree they expect, and earliestness, which ensures that a transducer produces its output as early as possible. Section 4.7 provide a semantic counterpart to the syntactic alignment. It lies the basis for a Myhill-Nerode type theorem for DTopI with top-down inspection. The minimal normal form that results of this Myhill-Nerode type theorem is presented in Section 4.8. Finally, Section 4.9 presents a sample-based learning algorithm of this normal form.

4.2 Illustration of Ideas

Before starting with a formalization, we illustrate informally, how we want to learn DTops from input-output examples. In a first step we need to find a Myhill-Nerode Theorem, i.e., that is unique normal form for DTops based on a semantic characterization. For this we will have to give a semantic justification to the unique normal forms for DTops proposed by Engelfriet, Maneth and Seidl [Engelfriet et al., 2009].

4.2.1 Myhill-Nerode Theorem for DTops

A DTop has rules of the form $q(f(x_1, \dots, x_k)) \to t$ which say that if the transducer is in state q and processes an input node labeled f, then it should output the tree t. The tree t is over output symbols, and may also contain "state calls" of the form $q'\langle x_i \rangle$ at its leaves. Such a call means to insert the result of translating in state q' the i-th subtree of the current input node. Thus, a DTop can be seen as a particular left-linear term rewriting system. Note that a variable x_i may occur many times in t ("copying"), or may not appear at all ("deletion"). If for every state q and input symbol f there is at most one rule, then the transducer is deterministic and realizes a partial function from trees to trees.

How can we define the analog to R-equivalence (mentioned before), for functions τ realized by DTops? Roughly speaking, we will chop τ into pieces, by considering functions from certain input subtrees to certain output subtrees. If there are only finitely many different such functions for τ, then τ can be realized by a DTOP. More precisely, consider an input tree s and a node π of s. The states of a DTOP that process the node π are uniquely determined by the *edge path* from the root of s to π (an edge path is the concatenation of pairs of node label and child-number on the path from the root to π). For a pair of edge paths $p = (u, v)$ we define the *residual* $p^{-1}\tau$ as all pairs (s', t') such that there are s, t with $\tau(s) = t$, u is an edge path in s to the subtree s', and v is an edge path in t to the subtree t'. Two pairs p_1, p_2 of edge paths are τ-equivalent if and only if $p_1^{-1}\tau = p_2^{-1}\tau$. Our Myhill-Nerode theorem says that for particular pairs of paths for input respectively output trees, τ-equivalence is of finite index if and only if τ can be realized by a DTOP.

4.2.2 Example Transformation

Let us consider an example. We want to exchange a list of A-nodes with a list of B-nodes. The lists are represented in the first-child-next-sibling encoding, while the empty list is represented by #. Thus, we want to transform

$$\mathsf{P}(\underbrace{\mathsf{A}(\#, \mathsf{A}(\#, \ldots \mathsf{A}(\#, \#) \ldots)}_{n}), \underbrace{\mathsf{B}(\#, \mathsf{B}(\#, \ldots \mathsf{B}(\#, \#) \ldots))}_{m})$$

into the tree obtained by exchanging the P's two subtrees.

$$\mathsf{P}(\underbrace{\mathsf{B}(\#, \mathsf{B}(\#, \ldots \mathsf{B}(\#, \#) \ldots)}_{m}), \underbrace{\mathsf{A}(\#, \mathsf{A}(\#, \ldots \mathsf{A}(\#, \#) \ldots))}_{n})$$

Since this transformation τ_{flip} is partial, there are exactly 4 different τ_{flip}-equivalence classes; the shortest representatives for these classes are the following pairs of paths for input respectively output trees:

$$\begin{aligned} q_1 &= (\varepsilon, \mathsf{P}1), \\ q_2 &= (\varepsilon, \mathsf{P}2), \\ q_3 &= (\mathsf{P}2, \mathsf{P}1), \\ q_4 &= (\mathsf{P}1, \mathsf{P}2) \end{aligned} .$$

These path pairs in this order correspond exactly to the states $q_1, \ldots, q_4$ of the unique minimal earliest uniform DTOP M_{flip} below. It starts with the axiom $\mathsf{P}(q_1\langle x_0\rangle, q_2\langle x_0\rangle)$ and has the following rules:

$$\begin{array}{lcl} q_1(\mathsf{P}(x_1, x_2)) & \to & q_3\langle x_2\rangle \\ q_2(\mathsf{P}(x_1, x_2)) & \to & q_4\langle x_1\rangle \\ q_3(\#) & \to & \# \\ q_3(\mathsf{B}(x_1, x_2)) & \to & \mathsf{B}(\#, q_3\langle x_2\rangle) \\ q_4(\#) & \to & \# \\ q_4(\mathsf{A}(x_1, x_2)) & \to & \mathsf{A}(\#, q_4\langle x_2\rangle) \end{array}$$

Note that a minimal earliest uniform DTOPs as defined in [Engelfriet et al., 2009] always comes together with a (minimal) deterministic top-down tree automaton recognizing the domain. In our example, consider the (q_4, A)-rule. It deletes the first subtree; without domain automaton this means that *any* tree would be accepted here, but we want only the tree # there.

Learning Algorithm

Using our Myhill-Nerode theorem for DTOPs, we show that for any given top-down tree transformation a *characteristic sample set* can be computed in polynomial time (with respect to the size n of the minimal DTOP). Given a characteristic sample set (or a superset), the learning algorithm correctly infers the desired transducer. The characteristic sample set for the example τ_{flip} of before consists of only four pairs of trees:

$$
\begin{array}{llcl}
[& \mathsf{P}(\#,\#) & / & \mathsf{P}(\#,\#), \\
 & \mathsf{P}(\mathsf{A}(\#,\#),\#) & / & \mathsf{P}(\#,\mathsf{A}(\#,\#)), \\
 & \mathsf{P}(\#,\mathsf{B}(\#,\#)) & / & \mathsf{P}(\mathsf{B}(\#,\#),\#), \\
 & \mathsf{P}(\mathsf{A}(\mathsf{A}(\#,\#),\#),\mathsf{B}(\mathsf{B}(\#,\#),\#)) & / & \mathsf{P}(\mathsf{B}(\mathsf{B}(\#,\#),\#),\mathsf{A}(\mathsf{A}(\#,\#),\#)) \quad]
\end{array}
$$

Note that a DTOP can transform a monadic input tree (of height n) into a full binary tree of height n. This implies that the trees in a characteristic sample set can have exponential size with respect to n. This can be avoided by representing output trees by their minimal DAGs; DAG representation of the output tree of a DTOP can be computed in linear time with respect to the size of the input tree (see [Maneth and Busatto, 2004]).

Inference of XML Transformations

XML documents are naturally modeled by *unranked trees*. There have been several proposal of tree transducers for unranked trees [Maneth and Neven, 1999, Maneth et al., 2005]. These models are more expressive than to use a classical ranked DTOP on the "first-child/next-sibling" (FC/NS) encoding of the unranked trees. For instance, consider the transformation XML_{flip} of a root node labeled P with n children labeled A followed by m children labeled B, into a root node with first the m B-nodes followed by the n A-nodes. This example can easily be realized by the unranked transducers of [Maneth and Neven, 1999, Maneth et al., 2005], however, *cannot* be realized by any ranked DTOP on FC/NS encoded trees. The reason is that a DTOP cannot change the order of nodes on a path.

Unfortunately, the added expressive power of unranked transducers comes at a price: we do not know whether deterministic unranked top-down tree transducers have decidable equivalence. In fact, since such transducers can completely flatten their output, they include the (classical) top-down tree-to-string translations. The equivalence problem for deterministic top-down tree-to-string transducers was recently proven to be decidable [Seidl et al., 2015] with a co-randomized polynomial algorithm for the linear case.

Are there other ranked tree encodings of unranked trees, so that a DTOP can realize XML_{flip}? We claim "yes". In fact, in the context of XML we believe that one should require the presence of input and output DTDs, before running the learning algorithm. We can use these DTDs to construct encodings that overcome restrictions of the FC/NS encoding. For instance, assume the following DTD for the input documents of XML_{flip}:

```
<!ELEMENT  P (A*,B*) >
<!ELEMENT  A EMPTY >
<!ELEMENT  B EMPTY >
```

And the same DTD, with A∗ and B∗ interchanged in the first line, for the output documents. Our idea of DTD-based encoding is to group elements from the same regular sub-expression, under a new tree node. In our example, we will have labels "(A*,B*)" (binary) and "A*", "B*" (unary). With this encoding, the input tree $\mathsf{P}(\mathsf{A},\mathsf{A},\mathsf{B})$ is represented as

$$\mathsf{P}(\text{“(A*,B*)”}(\text{“A*”}(\mathsf{A},\text{“A*”}(\mathsf{A},\text{“A*”}(\#,\#))),\text{“B*”}(\mathsf{B},\text{“B*”}(\#,\#)))$$

As we have seen before, a simple DTOP similar to M_{flip} can translate this tree into

$$\mathsf{P}(\text{“(B*,A*)”}(\text{“B*”}(\mathsf{B},\text{“B*”}(\#,\#)),\text{“A*”}(\mathsf{A},\text{“A*”}(\mathsf{A},\text{“A*”}(\#,\#))))).$$

Thus, if we supply adequately DTD-encoded trees to our learning algorithm, then it can infer a ranked transducer for XML_{flip}. This transducer has twelve states and sixteen rules, but can still be inferred by four examples, as for τ_{flip}. The transducer we obtain can, modulo syntax, be seen as an XSLT program for unranked trees, i.e., XML documents: rules correspond to `apply-templates` with the mode corresponding to the state. Note that the class of unranked tree transformations realized by DTOPs over DTD-encoded trees is strictly included in the unranked top-down translations of [Maneth and Neven, 1999, Maneth et al., 2005]; to see this, observe that the latter class contains both the DTD-encoding and the DTD-decoding.

4.3 Top-Down Tree Transducers

We study deterministic top-down tree transducers on ranked trees [Engelfriet, 1975], since these can be used to model a subclass of XSLT transformations [Maneth et al., 2005, Labath and Niehren, 2015], while being sufficiently restrictive to conserve some good algorithmic properties, such as decidability of equivalence [Engelfriet et al., 2009].

4.3.1 Top-Down Tree Transducers

We fix notations and present the standard definition of top-down tree transducers, together with some basic results.

We fix an infinite set $X = \{x_0, x_1, x_2, \dots\}$ of *input variables*, and, for every $k \geq 0$ define the set $X_k = \{x_1, \dots, x_k\}$, so that $X_0 = \emptyset$ in particular.

Definition 36. *A* deterministic top-down tree transducer *(*DTOP*) is a tuple* $M = (Q, F, G, Ax, rhs)$ *where:*

- Q *is a finite set of* states,

- *F and G are ranked alphabets of* input and output symbols*, respectively,*
- *$Ax \subseteq \mathcal{T}_G(Q \times \{x_0\})$ is a set with at most one element called the* axiom*,*
- *rhs is a partial function, which for any $k \geq 0$ maps elements from $Q \times F^{(k)}$ to trees in $\mathcal{T}_G(Q \times X_k)$.*

Note that the pair (q, x_i) will be noted $q\langle x_i\rangle$. We define the transformations $[\![M]\!]_q$ for all states q by mutual recursion and on induction of the size of the tree $s = f(s_1, \dots, s_k) \in \mathcal{T}_F$:

$$[\![M]\!]_q(f(s_1, \dots, s_k)) = rhs(q, f)\,[q'\langle x_i\rangle / [\![M]\!]_{q'}(s_i) \mid q' \in Q,\ 1 \leq i \leq k]$$

The transformation defined by *M is the partial function $[\![M]\!]$ from $\mathcal{T}_F$ to $\mathcal{T}_G$ such that for all $s \in \mathcal{T}_F$ for which the expression on the right is defined for some axiom $ax \in Ax$:*

$$[\![M]\!](s) = ax[q\langle x_0\rangle / [\![M]\!]_q(s) \mid q \in Q]$$

In the rest of the chapter, unless specified differently, F and G always denote (arbitrary) input and output alphabets, respectively. A DTop can be seen as a particular confluent and terminating term rewrite system, with left-linear rules. In fact, it is often intuitive to think of the rewrite rules that are induced by the family of right-hand sides of the transducer. If $t = rhs(q, f)$ for a DTop M, then $q(f(x_1, \dots, x_k)) \to t$ is called *the (q, f)-rule of M*.

As an additional remark, the particular case $Ax = \emptyset$, while necessary to provide stability properties, describes the empty transduction $\tau = \emptyset$. This case will often be ignored in this chapter, as for most of these proofs the empty case is trivial and cumbersome. We note $M = (Q, F, G, ax, rhs)$ (with ax instead of Ax) a transducer with exactly one axiom.

In the following two examples, we present two transducers that flip, copy, and delete subtrees.

Example 37. *We consider the transformation τ_{flip} which flips pairs of* A*-lists on the left and of* B*-lists on the right. An example for an input-output pair of τ_{flip} is given in Fig. 4.1. The signatures are $F = G = \{\mathsf{P}^{(2)}, \mathsf{A}^{(1)}, \mathsf{B}^{(1)}, \#^{(0)}\}$, where* P *is the pair constructor and # the end marker for lists. Transformation τ_{flip} can be defined by the* DTop *M_{37} with axiom $q_0\langle x_0\rangle$ and the following transitions:*

$$\begin{array}{llll}
(1) & q_0(\mathsf{P}(x_1, x_2)) \to \mathsf{P}(q_b\langle x_2\rangle, q_a\langle x_1\rangle) & & \\
(2) & q_a(\mathsf{A}(x_1)) \to \mathsf{A}(q_a\langle x_1\rangle) & (3) & q_a(\#) \to \# \\
(4) & q_b(\mathsf{B}(x_1)) \to \mathsf{B}(q_b\langle x_1\rangle) & (5) & q_b(\#) \to \#
\end{array}$$

The transducer M_{37} has axiom $q_0\langle x_0\rangle$, so it starts in state q_0 without producing any initial output. It then applies transition rule (1) to the root, which flips

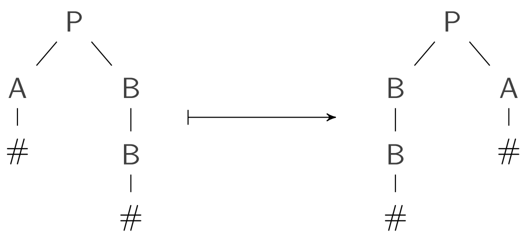


Figure 4.1: An input-output pair of transformation τ_{flip} flipping pairs of A-lists and B-lists.

its two children. From there, the state q_a copies any A*-list using rules (2) for* A*'s and (3) for #. Similarly, the state q_b copies any* B*-list using rules (4) for* B*'s and (5) for #.*

We next present an equivalent transducers that produces its output in an earlier manner, while copying and deleting instead of flipping subtrees.

Example 38. *The new* DTOP *M_{38} starts with the axiom* $\mathsf{P}(q_2\langle x_0\rangle, q_1\langle x_0\rangle)$ *and then applies the following transition rules:*

(0)	$ax = \mathsf{P}(q_2\langle x_0\rangle, q_1\langle x_0\rangle)$		
(1)	$q_1(\mathsf{P}(x_1, x_2)) \to q_a\langle x_1\rangle$	(2)	$q_2(\mathsf{P}(x_1, x_2)) \to q_b\langle x_2\rangle$
(3)	$q_a(\mathsf{A}(x_1)) \to \mathsf{A}(q_a\langle x_1\rangle)$	(4)	$q_a(\#) \to \#$
(5)	$q_b(\mathsf{B}(x_1)) \to \mathsf{B}(q_b\langle x_1\rangle)$	(6)	$q_b(\#) \to \#$

This transducer M_{38} outputs the root right away (possible as it is always labeled P*). The input tree is copied twice. One copy of the tree is read by q_2 which deletes its left son and send the right one to q_b using rule (2). The other is read by q_1 which deletes its right son and send the left one to q_a using rule (1). From there, as in M_{37} the state q_a copies any* A*-list using rules (3) for* A*'s and (4) for #. Similarly, the state q_b copies any* B*-list using rules (5) for* B*'s and (6) for #.*

4.3.2 Top-Down Domains

The domain of a transducer M is the set of input trees s for which $[\![M]\!](s)$ is well-defined, i.e., $dom[\![M]\!]$. It is folklore [Engelfriet, 1977] that the domain of any DTOPs is accepted by some DTTA, as restated in the following lemma.

Lemma 39. *The domain of any* DTOP *is recognizable by some* DTTA*.*

Proof. Let $M = (Q, F, G, \{ax\}, rhs)$ be a DTOP and s an input tree (note that if M has no axiom, $dom([\![M]\!]) = \emptyset$, which is obviously recognized by a DTTA). Intuitively, s is in $dom([\![M]\!])$ if M has an axiom, and at no point during the construction of $[\![M]\!](s)$ do we call $[\![M]\!]_q$ on a subtree $u^{-1}s$ such that $s[u] = f$, but $rhs(q, f)$ is undefined. However, unlike for DTTAs, a DTOP may visit the same subtree $u^{-1}s$ several times and in different states during a computation. In this case, such a subtree must belong to the domain $dom(\llbracket M \rrbracket_q)$ for all such q. Therefore, we need to reason about the subsets of states of M that visits $u^{-1}S$.

We define the automaton $A = (F, P, p_I, \Delta)$ recognizing $dom(M)$ as follows.

- The states $P = \{Q' \mid Q' \subseteq Q\}$.
- The initial state is $p_I = Q_I = \{q \mid q\langle x_0 \rangle$ occurs in $ax\}$.
- For $Q' \in P$, and $f \in F^{(k)}$ such that for all $q \in Q'$, $rhs(q, f)$ is defined, then the rule of Δ on (Q', f) is $Q' \xrightarrow{f} (Q_1, \ldots, Q_k)$, where for all i from 1 to k, $Q_i = \bigcup_{q \in Q'} \{q' \mid q'\langle x_i \rangle$ occurs in $rhs(q, f)\}$.

We next prove that $\llbracket A \rrbracket_{Q'} = \cap_{q \in Q'} dom(\llbracket M \rrbracket_q)$. This is done by induction on the input tree s. For $s = f(s_1, \ldots, s_k)$, we have $s \in \cap_{q \in Q'} dom(\llbracket M \rrbracket_q)$ if and only if for every $q \in Q'$, $rhs(q, f)$ is defined, and for every $q'\langle x_i \rangle$ occuring in $rhs(q, f)$, $s_i \in dom(\llbracket M \rrbracket_{q'})$. In other words, for every $q \in Q'$, $rhs(q, f)$ is defined, and for every i from 1 to k, for all q' such that $q'\langle x_i \rangle$ occurs in a $rhs(q, f)$, $q \in Q'$, then $s_i \in dom(\llbracket M \rrbracket_{q'})$. For every i, these q' describe exactly the set Q_i in our construction such that $Q' \xrightarrow{f} (Q_1, \ldots, Q_k)$. Since by recursion, $s_i \in \cap_{q' \in Q_i} dom(\llbracket M \rrbracket_{q'})$ is the same as $s_i \in \llbracket A \rrbracket_{Q_i}$, we have that $s \in \cap_{q \in Q'} dom(\llbracket M \rrbracket_q)$ if and only if for all $q \in Q'$, $rhs(q, f)$ is defined, and for i from 1 to k, $s_i \in \llbracket A \rrbracket_{Q_i}$. Since $Q' \xrightarrow{f} (Q_1, \ldots, Q_k)$ is a rule of Δ, so this is equivalent to $s \in \llbracket A \rrbracket_{Q'}$.

From there, we justify the choice of Q_I as an initial state. By definition of $\llbracket M \rrbracket$, $s \in dom(\llbracket M \rrbracket)$ if and only if for all q such that $q\langle x_0 \rangle$ occurs in ax, then $s \in dom(\llbracket M \rrbracket_q)$. This is equivalent to say that $s \in \bigcap_{q \in Q_I} dom(\llbracket M \rrbracket_q)$ which, as seen above, is equivalent to $s \in \llbracket A \rrbracket_{Q_I}$. Hence, $\llbracket A \rrbracket = \llbracket A \rrbracket_{Q_I} = dom(\llbracket M \rrbracket)$. □

4.3.3 Domain Inspection

We note an important weakness of DTOPs: they are not closed under domain restrictions by DTTAs, since they cannot traverse of check those subtrees of the input tree, that do not produce any output.

Example 40. *The finite partial function $\tau_{40} = [f(c, a)/a, f(c, b)/b]$ cannot be defined by any* DTOP. *This problem is that any* DTOP *must produce the output at the second leaf of the input trees, since the constant that is output depends on which is this leaf. And since nothing else may be output, nothing may be output at the first leaf. Therefore, the first subtree of the input tree cannot be traversed by any* DTOP *defining τ_{40}, so it cannot be checked whether the first subtree is a c leaf. Nevertheless, there exists a* DTOP *M_{40} such that* *if $s \in dom(\tau_{40})$, then $[\![M_{40}]\!](s) = \tau_{40}(s)$. M_{40} has two states q_1, q_2, an axiom $q_0\langle x_0\rangle$ and the following transition rules:*

$$(1) \quad q_0(f(x_1,x_2)) \to q_1\langle x_2\rangle \qquad (2) \quad q_1(a) \to a \qquad (3) \quad q_1(b) \to b$$

Futhermore, $dom(\tau_{40}) = \{f(c,a), f(c,b)\}$ is DTTA*-definable. Therefore, this example shows that the class of* DTOPs *is not closed under top-down inspection, i.e., by domain inspection with* DTTAs.

In order to resolve this problem, we follow the approach of [Engelfriet et al., 2009, Lemay et al., 2010], and extend DTOPs with domain inspection. In contrast to there, however, we will not only consider domain inspection by DTTAs, but permit more general devices for defining tree languages.

Definition 41. *A* DTOPI *is a* DTOP *with domain inspection, i.e., a pair $N = (M, D)$ where M is a* DTOP *with input signature F and $D \subseteq \mathcal{T}_F$ a set of input trees.*

The semantics of a DTOPI is defined by domain restriction, i.e., $[\![N]\!] = [\![M]\!]_{|D}$. Claerly, $dom([\![N]\!]) = dom([\![M]\!]) \cap D$. Note that we admit nonregular tree languages D as inspection domains, so that the domain of the transformation of a DTOPI may be nonregular too.

Definition 42. *A* DTOPI$_{reg}$ *is a* DTOPI *whose inspection domain is regular and a* DTOPI$_{td}$ *is a* DTOPI *whose inspection domain is recognizable by some top-down deterministic tree automata (i.e. it is regular and path-closed).*

Example 43. *We can define the transformation τ_{flip} by the* DTOPI *$N_{43} = (M_{43}, dom(\tau_{flip}))$ such that M_{43} flips arbitrary pairs, and not only pairs of* A*-list and* B*-lists as done by the* DTOPs *M_{37} and M_{38} from Examples 37 and 38. For this, a single state q is sufficient. Furthermore, M_{43} has the axiom $q\langle x_0\rangle$ and the following transition rules:*

$$\begin{array}{llll} (1) & q(\mathsf{P}(x_1,x_2)) \to \mathsf{P}(q\langle x_2\rangle, q\langle x_1\rangle) & (2) & q(\mathsf{A}(x_1)) \to \mathsf{A}(q\langle x_1\rangle) \\ (3) & q(\mathsf{B}(x_1)) \to \mathsf{B}(q\langle x_1\rangle) & (4) & q(\#) \to \# \end{array}$$

Clearly, $dom(\tau_{flip})$ is strictly subsumed by $dom(M_{43})$, so that external domain inspection is needed to define τ_{flip} properly with these more generic rules.

Proposition 44. *The domain and inspection domain of any* DTOPI *$N = (M, D)$ are related as follows:*

- *if D is regular then $dom([\![N]\!])$ is regular*

- *if D is path-closed then $dom([\![N]\!])$ is path-closed*

So if D is definable by some DTTA *then $dom([\![N]\!])$ is definable some* DTTA *too.*

Proof. Lemma 39 shows that $dom([\![M]\!])$ is always definable by a DTTA, and thus path-closed and regular. Therefore, the Proposition follows from $dom([\![N]\!]) = dom([\![M]\!]) \cap D$, and the fact that both path-closedness and regularity are closed under intersection. □

4.4 Syntactic Equivalence

We introduce a notion of syntactic alignment computed by DTopIs, that relate the paths of input trees to the paths of output trees that they produce. We then define a equivalence relation on syntactically aligned pairs of paths, stating that the DTopIs performs the same transformation there.

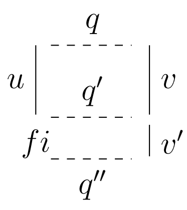
4.4.1 Syntactic Alignment

We define a notion of syntactic alignment for transducers, to track which paths of output trees are produced by which paths of input trees.

The judgements $u \sim_{q'}^{q} v$ and $u \sim_{q} v$ we aim to define describe which input subtrees produce which output subtrees in which states for a DTopI N. Pairs$Succ_N(u,v)$ can thus be understood as the pairs at the "next step" in the computation of an image $[\![N]\!](s)$, where $s \models u$.

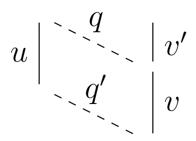
Definition 45. *Let $N = (M, D)$ be a* DTopI*. We define judgements $u \sim_{q'}^{q} v$ stating that an input path u is aligned to an output path v in state q' when starting from state q, by the following inferences rules where $f^{(k)} \in F$ and $1 \leq i \leq k$:*

$$\frac{true}{\varepsilon \sim_{q}^{q} \varepsilon} \qquad \frac{u \sim_{q'}^{q} v \qquad rhs(q', f) \models v'q''\langle x_i \rangle}{ufi \sim_{q''}^{q} vv'}$$



Furthermore, we define judgements $u \sim_{q'} v$, stating that u is syntactically aligned to v in state q' (or that the pair (u,v) is syntactically aligned in state q'), when starting with the axiom:

$$\frac{ax \models v'q\langle x_0 \rangle \qquad u \sim_{q'}^{q} v}{u \sim_{q'} v'v}$$



Note that the notion of syntactic alignment for $N = (M, D)$ depends only of the DTop M, so it is independent of the domain inspection D.

Example 46. *We reconsider the two transducers defining the function τ_{flip}. For transducer M_{37} from Example 37, which was not earliest, the pair $\varepsilon \sim_{q_0} \varepsilon$, $\mathsf{P1} \sim_{q_b} \mathsf{P2}$, and $\mathsf{P2} \sim_{q_a} \mathsf{P1}$. For the earliest transducer M_{38} from Example 38, we have $\varepsilon \sim_{q_b} \mathsf{P1}$, $\varepsilon \sim_{q_a} \mathsf{P2}$, $\mathsf{P1} \sim_{q_a} \mathsf{P2A1}$, and $\mathsf{P1B1} \sim_{q_b} \mathsf{P2B1}$.*

We define the notion of *syntactic successors* of a syntactically aligned pair p in a top-down manner.

Definition 47. *Let $N = (M, D)$ a DTopI, $M = (Q, F, G, ax, rhs)$, $p = (u, v)$ a pair syntactically aligned in a state $q \in Q$. For any $f \in F$ such that $rhs(q, f)$ is defined, and any path v' such that $v'^{-1}rhs(q, f) = q'\langle x_i\rangle$ for some $q' \in Q$ we define:*

- *$ind_N(p, f, v') = i$ the index of p, f, v' and*
- *$succ_N(p, f, v') = (ufi, vv')$ the successor of p, f, v'.*

Furthermore, we define $Succ_N(p)$ as the set of all syntactic successors of p with respect to N and some f, v', i.e.:

$$Succ_N(p) = \{succ_N(p, f, v') \mid f \in F,\ q, q' \in Q,\ v'^{-1}rhs(q, f) = q'\langle x_i\rangle\}$$

Corollary 48. *Let $N = (M, D)$ a DTopI, $p = (u, v)$ a pair syntactically aligned in a state $q \in Q$. Then all pairs of $Succ_N(p)$ are syntactically aligned in some state $q' \in Q$.*

Proof. Let $M = (Q, F, G, ax, rhs)$. Since $p = (u, v)$ is syntactically aligned in $q \in Q$, there exists v_0, v_1 and a state $q_0 \in Q$ such that $v = v_0v_1$, $u \sim_q^{q_0} v_1$ and $v_0^{-1}ax = q_0\langle x_0\rangle$. A pair $(ufi, vv') = succ_N(p, f, v')$ if $rhs(q, f)$ is defined, and $v'^{-1}rhs(q, f) = q'\langle x_i\rangle$. This means that $ufi \sim_{q'}^{q_0} v_1v'$, and thus $ufi \sim_{q'} vv'$. $\square$

The following proposition states the general relevance of syntactic alignment for the transformation defined by a transducer.

Proposition 49. *Let $N = (M, D)$ be a DTopI, q a state of M, and (u, v) a pair of input-output paths. If $u \sim_q v$ and $s \in dom([\![N]\!])$ is an input tree with $s \models u$, then $v^{-1}([\![N]\!](s)) = [\![M]\!]_q(u^{-1}s)$.*

Proof. We first show for all u, v, q, q'' that if $u \sim_{q''}^{q} v$ then any $s \in dom([\![M]\!]_q)$ with $s \models u$ satisfies $v^{-1}([\![M]\!]_q(s)) = [\![M]\!]_{q''}(u^{-1}s)$. The proof is by induction on the definition of judgements $u \sim_{q''}^{q} v$.

- In the first case, the judgement $u \sim_{q''}^{q} v$ is derived by the initial rule:

$$\frac{true}{\varepsilon \sim_{q}^{q} \varepsilon}$$

  Hence, $u = v = \varepsilon$ and $q = q''$. The output tree $[\![M]\!]_q(s)$ is trivially well-defined for all $s \in dom([\![M]\!]_q)$. Furthermore, it is equal to $\varepsilon^{-1}([\![M]\!]_q(s)) = [\![M]\!]_q(\varepsilon^{-1}s)$.

- In the second case, the syntactic alignment $u \sim_{q''}^{q} v$ is defined as follows, where $u = u'fi$, $f^{(k)} \in F$, $1 \leq i \leq k$, and $v = v'v''$:

$$\frac{u' \sim_{q'}^{q} v' \qquad rhs(q', f) \models v''q''\langle x_i \rangle}{u'fi \sim_{q''}^{q} v'v''}$$

  The induction hypothesis applied to $u' \sim_{q'}^{q} v'$ shows that $v'^{-1}[\![M]\!]_q(s) = [\![M]\!]_{q'}(u'^{-1}s)$. Since $s \models u$, we have $u'^{-1}s = f(s_1, \ldots, s_k)$ for some $s_1, \ldots, s_k$. The recursive definition of $[\![M]\!]_{q'}$ yields:

$$[\![M]\!]_{q'}(u'^{-1}s) = rhs(q', f)\,[\tilde{q}\langle x_j \rangle \leftarrow [\![M]\!]_{\tilde{q}}(s_j) \mid \tilde{q} \in Q,\ 1 \leq j \leq k]\,.$$

  This gives us $v''^{-1}[\![M]\!]_{q'}(u'^{-1}s) = [\![M]\!]_{q''}(s_i)$. Therefore:

$$v^{-1}[\![M]\!]_q(s) = v''^{-1}[\![M]\!]_{q'}(u'^{-1}s) = [\![M]\!]_{q''}(s_i) = [\![M]\!]_{q''}(u^{-1}s)$$

In order to prove the proposition, we recall that for any $s \in dom([\![N]\!])$:

$$[\![N]\!](s) = [\![M]\!](s) = ax[\tilde{q}\langle x_0 \rangle \leftarrow [\![M]\!]_{\tilde{q}}(s) \mid \tilde{q} \in Q].$$

The judgement $u \sim_{q} v$ must be derived as follows where $v = v'v''$:

$$\frac{ax \models v'q\langle x_0 \rangle \qquad u \sim_{q'}^{q} v''}{u \sim_{q'} v'v''}$$

Since $ax \models v'q\langle x_0 \rangle$, we have $v^{-1}N(s) = v''^{-1}[\![M]\!]_q(s)$. From $u \sim_{q'}^{q} v''$ the above claim shows $v''^{-1}[\![M]\!]_q(s) = [\![M]\!]_{q''}(u^{-1}s)$ and thus $v^{-1}N(s) = [\![M]\!]_{q''}(u^{-1}s)$. $\square$

4.4.2 Trimmed Transducers

The notion of syntactic alignments leads to a proper notion of trimmed DTopIs:

Definition 50. *A* DTopI *N is called trimmed if it does not contain any useless states and rules, where:*

- *a state q of N is called useless if there is no pair $p = (u, v)$ aligned in q such that $s \models u$ for some tree $s \in dom([\![N]\!])$, and*

- *a transition $rhs(q, f)$ is called useless if there exists no pair (u, v) aligned in q such that $s \models uf$ for some $s \in dom([\![N]\!])$.*

A trimmed version of a DTopI N is the DTopI $trim(N)$ that is obtained from N by removing useless states, useless transitions, and all terms with useless states from the set of axioms.

Lemma 51. *Any* DTopI$_{reg}$ *is equivalent to some trimmed* DTopI$_{reg}$ *and any* DTopI$_{td}$ *is equivalent to some trimmed* DTopI$_{td}$.

Proof. If N is a DTopI$_{\text{reg}}$ then $trim(N)$ is an equivalent trimmed DTopI$_{\text{reg}}$, and if N is a DTopI$_{\text{td}}$ then $trim(N)$ is an equivalent trimmed DTopI$_{\text{td}}$. □

The computation of $trim(N)$ from N requires to identify the useless states and rules of N, which is less obvious. The actual construction is given in the appendix.

4.4.3 Origins of Output Constructors

The notion of syntactic alignment allows to define the "origin" of any constructor of an output tree produced by a DTop, i.e., the unique path of the input tree, at which the DTop produced that label.

Proposition 52. *Let M be a* DTop *and $s \in dom(M)$ an input tree. Then for any output path vg such that $[\![M]\!](s) \models vg$, either $ax \models vg$ or there exist a unique decomposition $v = v'v''$, an input path $u'f$, and a state q' such that:*

- *$s \models u'f$,*

- *$u' \sim_{q'} v'$, and*

- *$rhs(q', f) \models v''g$.*

The intuition is that each constructor of the output tree is created by the DTop in a production step at a unique input node, where the correspondence between input and output nodes is captured by the notion of syntactic alignments.

The formal proof is given in the appendix. It is not difficult but a little cumbersome since it requires an equivalent bottom-up definition of syntactic alignments.

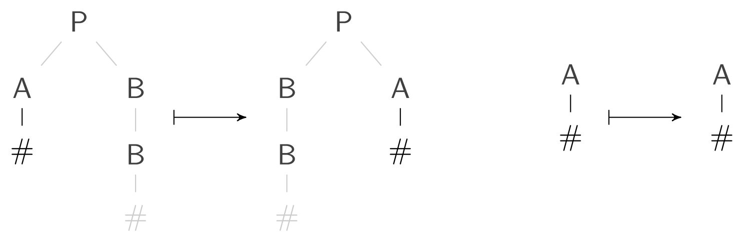


Figure 4.2: A pair of τ_{flip} on the left, and a pair of its residual at $(\mathsf{P}1, \mathsf{P}2)$ on the right.

4.4.4 Syntactic Equivalence

Residuals play a central role in Myhill-Nerode theorems, as known from the cases of deterministic finite word automata [Nerode, 1958] and of subsequential transducers [Oncina et al., 1993]. Therefore, we would like to define a notion of *residuals* of tree transformations, independent of the transducers that might compute it, that state what transformation remains to be done at the current "event" of a top-down transduction process. Such events are pairs of paths $p = (u, v)$, stating that path u of the input tree was read, for producing the output tree until path v.

Definition 53. *The* residual $p^{-1}\tau$ *of a partial function* $\tau \subseteq \mathcal{T}_F \times \mathcal{T}_G$ *at a pair* $p = (u, v)$ *of an* F*-path and a* G*-path, is the relation* $p^{-1}\tau \subseteq \mathcal{T}_F \times \mathcal{T}_G$ *with:*

$$p^{-1}\tau = \{(u^{-1}s, v^{-1}t) \mid (s,t) \in \tau,\ s \models u, t \models v\} \ .$$

Example 54. *For the transformation* τ_{flip} *and the pair of path* $p = (\mathsf{P}1, \mathsf{P}2)$*, the residual* $p^{-1}(\tau_{\mathit{flip}})$ *is the identity on* A*-lists (see Fig 4.4.4).*

In general, each transformation can have an infinite number of different residuals for the infinitely many possible pairs p of paths. However, we only consider very particular pairs of paths. For instance, we do not care about p's such that $p^{-1}\tau$ is not a function. This happens if the node v was generated by an input subtree that is disjoint (i.e., in a different subtree) with u. For example, for τ_{flip}, the residual of $(\mathsf{P}1, \mathsf{P}1)$ is not functional. We also do not care about pairs $p = (u, v)$ for which the residual $p^{-1}\tau$ is empty. This happens if u does not belong to any input tree $s \in \mathit{dom}(\tau)$, or if v is not a node of any $\tau(s)$ where $s \in \mathit{dom}(\tau)$. For example, for τ_{flip}, this happens for the pairs with $u = \mathsf{P}1\mathsf{B}1$ or $v = \mathsf{P}1\mathsf{A}1$.

The next lemma shows for any DTopI $N = (M, D)$ that if a pair of paths p is aligned to q in a transducer N as in Definition 45 then $p^{-1}[\![N]\!]$ is a partial function depending on state q and on the residual of the domain.

Lemma 55. *Let* $N = (M, D)$ *be a* DTopI *with state* q*. If* $p = (u, v)$ *satisfies the syntactic alignement* $u \sim_q v$*, then* $p^{-1}[\![N]\!] = [\![M]\!]_{q|u^{-1}\mathit{dom}([\![N]\!])}$.

Proof. Proposition 49 gives us $v^{-1}[\![N]\!](s) = [\![M]\!]_q(u^{-1}s)$. By definition of $p^{-1}[\![N]\!]$, $v^{-1}[\![N]\!](s) = p^{-1}[\![N]\!](u^{-1}s)$. We then have $p^{-1}[\![N]\!](u^{-1}s) = [\![M]\!]_q(u^{-1}s)$ for all $s \in dom([\![N]\!])$. By definition, $u^{-1}dom([\![N]\!]) = \{u^{-1}s \mid s \in dom([\![N]\!]), s \models u\}$. We then have $p^{-1}[\![N]\!](t) = [\![M]\!]_q(t)$ for all $t \in u^{-1}dom([\![N]\!])$. □

Definition 56. *Let $N = (M, D)$ be a* DTopI. *We define the* congruence relation *$\equiv_N$ on pairs p_1 and p_2 of labeled paths that are syntactically aligned by N as follows:*

$$p_1 \equiv_N p_2 \text{ iff } p_1^{-1}[\![N]\!] = p_2^{-1}[\![N]\!]$$

Corollary 57. *The syntactic congruence $\equiv_N$ of a* DTopI$_{reg}$ *$N = (M, D)$ has finite index.*

Proof. Let $p = (u, v)$ be a pair of paths such that $u \sim_q v$. Lemma 55 implies that $p^{-1}[\![M]\!] = [\![M]\!]_{q|u^{-1}dom([\![N]\!])}$. The domain $dom([\![N]\!])$ is $dom([\![M]\!]) \cap D$. Since both sets are regular it follows that $dom([\![N]\!])$ is regular to. Let A be a trimmed nondeterministic tree automaton (with state set R) that recognizes $dom([\![N]\!])$. Lemma 14 shows that $u^{-1}dom([\![N]\!]) = \cup_{r \in R_u}[\![A]\!]_r$. Hence, $p^{-1}[\![M]\!]$ is characterized by a state q of M and a subset R_u of states of A. Since there are finitely many choices for both, there exists only finitely many possible values of $p^{-1}[\![M]\!]$ for all aligned paths p. □

This corollary is a kind of Myhill-Nerode theorem, but has the disadvantage that the congruence relation $\equiv_N$ is defined on objects that depend on the transducer N, rather than only on the transformation $[\![N]\!]$. Therefore, it does not immediately lead us to a unique minimal normal form of the transformation. For example, the two transducers presented for τ_{flip} in Examples 37 and 38 both have no redundant states, but their equivalences are incomparable: neither is a refinement of the other.

4.5 Compatible Transducers

Equivalent DTopIs may check the membership of an input tree to the domain of the transformation in many different manners. In the one extreme case, where no output is to be produced, the job can be entirely done by the domain inspection. In the other extreme case, the domain can be entirely checked by the underlying DTop. In general case, the DTop and the domain inspection have to share the job in some way or another.

Example 58. *We first consider a* DTopI *which mostly leaves the membership test of the input tree to the domain inspection. It is the* DTopI *$(M_{43}, dom(\tau_{flip}))$ from Example 43. This* DTopI *defines the transformation τ_{flip} which flips any pair of* A*-lists and* B*-lists. Its* DTop *$(M_{43}, dom(\tau_{flip}))$ has* *a single state q and the pairs* $(\mathsf{P1}, \mathsf{P2})$ *and* $(\mathsf{P2}, \mathsf{P1})$ *are both aligned in q. Even though aligned in the same state, the residuals of these pairs differ in their domains. The domain of the residual* $(\mathsf{P1}, \mathsf{P2})^{-1}\tau$ *is the set of* A*-lists, that is* $\mathsf{P1}^{-1}D$ *where* $D = dom(\tau_{flip})$*, while the domain of the residual* $(\mathsf{P2}, \mathsf{P1})^{-1}\tau$ *is the set of all* B*-lists, that is* $\mathsf{P2}^{-1}D$*.*

This example illustrates that the residual of a pair $p = (u, v)$ aligned in state q by a DTopI $N = (M, D)$ may still depend on $u^{-1}dom([\![N]\!])$ as stated in Lemma 55, and not only on $[\![M]\!]_q$ and D as one might hope for. In a canonical DTopI this should not be the case.

Definition 59. *We call a* DTopI $N = (M, D)$ *compatible if* $D = dom([\![N]\!])$*, and if* $dom(p^{-1}[\![N]\!])$ *coincides for all pairs p that are syntactically aligned in the same state of N.*

The notion of compatible DTopI is a semantic counterpart of the syntactic notion of *uniform* DTopI in [Engelfriet et al., 2009]. It is not only much simpler but also more general: While the notion of uniform DTopI depends on the Dtta that defines the inspection domain and is thus restricted to top-down inspection, the notion of compatibility applies to general DTopI.

Each state of a compatible DTopI N indeed corresponds to an equivalence class of the syntactic equivalence $\equiv_N$, which is determined by the state to which the pairs in this equivalence class are aligned:

Lemma 60. *Let N be a compatible* DTopI*. If two pairs p and p' are syntactically aligned in the same state, then $p \equiv_N p'$.*

Proof. Let $p = (u, v)$ and $p' = (u', v')$ be both syntactically aligned in the same state of N, say q. Lemma 55 then shows that $p^{-1}[\![N]\!] = [\![M]\!]_{q|u^{-1}dom([\![N]\!])}$ and $p'^{-1}[\![N]\!] = [\![M]\!]_{q|u'^{-1}dom([\![N]\!])}$. By compatibility, the residuals of the domain are the same: $u^{-1}dom([\![N]\!]) = u'^{-1}dom([\![N]\!])$. Therefore, the residuals of the transducer are the same: $p^{-1}[\![N]\!] = p'^{-1}[\![N]\!]$, that is $p \equiv_N p'$. □

We next show that any $\text{DTopI}_{\text{reg}}$ can be made compatible. The intuition is that a compatible transducer should check as many domain restrictions as possible by itself, rather than delegating this job to the domain inspection. In order to do so, it should run in parallel with its DTop some Dtta that tests membership to the path-closure of the inspection domain, i.e., to the least path-closed tree language subsuming the inspection domain

Example 61. *We reconsider the* DTop M_{43} *from Example 58 which defines* τ_{flip}*. When making* M_{43} *compatible, we will obtain the* DTopI (M_{37}, D) *where* $D = dom(\tau_{flip})$*. The single state q of M_{43} will be split into the* 3 *different states* q_0*,* q_a *and* q_b *of* M_{37}*. In order to see how this works, we consider the following top-down tree automaton A recognizing D:*

$$p_0 \xrightarrow{\mathsf{P}} (p_a, p_b) \qquad p_a \xrightarrow{\mathsf{A}} (p_a) \qquad p_a \xrightarrow{\#} () \qquad p_b \xrightarrow{\mathsf{B}} (p_b) \qquad p_b \xrightarrow{\#} ()$$

Note that this tree automaton is top-down deterministic, which simplifies the example a little bit. The general construction, however, can be done also be lifted to nondeterminstic top-down tree automata recognizing D. It should also be noticed that the result of the construction depends of which tree automaton was chosen.

The states of the compatible DTopI *that we obtain with* Dtta A *will be the pairs of a state of* M_{37} *and a state of* A*, that is the pairs* $q_0 = (q, p_0)$*,* $q_a = (q, p_a)$*, and* $q_b = (q, p_b)$*. The transition rules will be obtained by pairing transitions of the* DTop *of* M_{37} *and the tree automaton* A *in the obvious manner. Indeed, the resulting* DTopI *is* (M_{37}, D)*, which is compatible.*

We next prove that any $\mathrm{DTopI}_{\mathrm{reg}}$ or $\mathrm{DTopI}_{\mathrm{td}}$ can be made compatible.

Proposition 62. *There exists an algorithm that given a* DTop M *and a top-down tree automaton* A *computes in time* $O(|M|\ 2^{|M|+|A|})$ *a compatible* DTopI *equivalent to the* DTopI $(M, [\![A]\!])$*. Furthermore, if* A *was top-down deterministic then the resulting* DTopI *is a* DTopI_{td}*.*

Proof. Let $D = [\![A]\!]$, $N = (M, D)$ a $\mathrm{DTopI}_{\mathrm{reg}}$, and $D' = dom([\![N]\!])$ By Lemma 39, we can construct in time $O(|A|\ 2^{|M|})$ a top-down tree automaton A' that recognizes D'. Note that if A is top-down deterministic, then A' is as well. In the general case, however, A' may be nondeterministic. This is a problem since it may be impossible to run A deterministically in a top-down manner, so that no DTop may not be able to check membership to D' exactely. What a DTop may still do is to compute at any path the set of states that A' reaches, while ignoring the dependencies between the states of siblings.

The first idea is to build a DTop M' that runs M while computing the set of reachable states of A' in parallel. The states of M' are pairs (q, P) where q is a state of M and P is a subset of states of A'. The axiom of M' is obtained from the axiom of M by replacing $q\langle x_0\rangle$ by $(q, P_I)\langle x_0\rangle$ where P_I is the set of initial states of A. The rules $rhs'((q, P), f)$ are obtained from the rules $rhs(q, f)$ of M, by replacing any leaf of the form $q'\langle x_i\rangle$ for some q' by $(q', P')\langle x_i\rangle$, where $P' = \{p_i \mid p \in P,\ p \xrightarrow{f} (p_1, \ldots, p_i, \ldots, p_n) \text{ a rule of } A'\}$. It is not difficult to see that $N' = (M', D')$ is equivalent to N. We now argue that N' is compatible. We claim that if $u \sim_{(q,P)} v$ in M' then P is the set of states reached by A over u, starting at P_I in the axiom and progressing step by step in the rules. Hence, in this case we have $u^{-1}D' = u^{-1}[\![A']\!] = \cup_{p\in P}[\![A']\!]_p$ by Lemma 14. This shows that the dependence of the residual $u^{-1}D'$ on u is limited to a dependence on P and thus on the state of M' to which (u, v) is aligned. So if also $u' \sim_{(q,P)} v'$ then $u'^{-1}D' = \cup_{p\in P}[\![A']\!]_p$ and thus $u^{-1}D' = u'^{-1}D'$ as required.

However, the construction of M' may require double-exponential time, since it requires exponential time in the size of A', which itself may be exponential in the size of M. We thus need to improve the construction. For this we note that $D' = [\![A]\!] \cap [\![A'']\!]$ where A'' is the tree automata that recognizes $dom([\![M]\!])$ from Lemma 39. We note that A'' is top-down deterministic and of size at most $O(2^{|M|})$. Unfortunately, we cannot always make A top-down deterministic. So, rather than computing states reachable by the intersection of A and A'', the second idea is sufficient to compute the unique state reached by A'' and the subset of states reached by A. We thus construct a DTopI M'' that runs M in parallel with computing the reachable states of A and the state reached by A''. The states of M'' are thus triples (q, p'', P) where q is a state of M, p'' is a state of A'' and P a subset of states of A. The construction of M'' can be done similarly to before but now in time $O(|M|\ |A''|\ 2^{|A|})$. Clearly the DTopI (M'', D') is equivalent to (M', D'), and it is not difficult to see that it is compatible too. □

Corollary 63. *Any* DTopI *is equivalent to some trimmed compatible* DTopI.

Proof. This follows from Proposition 62, since for any compatible DTopI N, the DTopI $trim(N)$ is compatible too, and trivially trimmed. □

4.6 Earliest Transducers

We introduce earliest transducers in order to normalize the output production and thereby to find some kind of unique minimal transducers for a given transformation. The idea is to produce the output as early as possible, as first proposed for subsequential transducers by Choffrut [Choffru, 1978, Choffrut, 2003] and extended to any DTopI$_{\text{td}}$ by Engelfriet, Maneth and Seidl [Engelfriet et al., 2009]. Our approach is yet more general in that it applies to any DTopI$_{\text{reg}}$, i.e. we capture regular domain inspection in addition. This generalization requires a more flexible notion of earliest DTopIs, that is independent from the notion of uniform transducers. We do so by considering the output production of aligned pairs (here the inspection domain intervenes), and not only the production of the state to which the pair is aligned.

We have to define what it means for a DTopI to maximize its output production. The definition will be based on the notion of largest common tree prefixes. For two trees $t, t' \in \mathcal{T}_G$ we define their *largest common prefix tree* $t \sqcap t' \in \mathcal{T}_G(\{\perp\})$ as follows:

$$g(t_1, \ldots, t_k) \sqcap g'(t'_1, \ldots, t'_{k'}) = \begin{cases} g(t_1 \sqcap t'_1, t_2 \sqcap t'_2, \ldots, t_k \sqcap t'_k) & \text{if } g = g' \\ \perp & \text{otherwise.} \end{cases}$$

The $\sqcap$ operator is associative and commutative, so that it can be easily lifted to finite sets of trees $D = \{t_1, \ldots, t_n\}$, by defining $\bigsqcap D = t_1 \sqcap t_2 \sqcap \cdots \sqcap t_n$

independently of the ordering of the trees in D.

Let τ be a partial function and u a input path u in $paths(dom(\tau))$. We define *τ's maximal output at u* as:

$$out_\tau(u) = \bigsqcap\{\tau(s) \mid s \models u, s \in dom(\tau)\}$$

For any partial function $\tau \neq \emptyset$, we call $out_\tau(\varepsilon)$ the "global common prefix" of the range of τ. Similarly, we define the maximal output at a npath uf by $out_\tau(uf) = \bigsqcap\{\tau(s) \mid s \models uf, s \in dom(\tau)\}$. Note that $out_\tau(u)$ is undefined if there does not exist any tree $s \in dom(\tau)$ such that $s \models u$.

Example 64. *For τ_{flip}, $out_{\tau_{flip}}(\varepsilon) = f(\bot, \bot)$, as every tree in the range of τ_{flip} has the form $f(s_1, s_2)$ for some input trees s_1 and s_2. For the input path $u = f1a1a$, we have $out_{\tau_{flip}}(u) = f(\bot, a(a(\bot)))$, since all inputs having this path u must be of the form $f(a(a(s_1), s_2)$ for some input trees s_1 and s_2.*

We now consider *earliest* transducers, that always produce output constructors as soon as possible.

Definition 65. *A* DTopI *$N = (M, D)$ is* earliest *if for any pair p that is syntactically aligned by M, the residual $p^{-1}[\![N]\!]$ satisfies $out_{p^{-1}[\![N]\!]}(\varepsilon) = \bot$.*

Example 66. *We reconsider the transducers defining τ_{flip}. It can be defined by the* DTop *M_{38} which is earliest, and by the* DTop *M_{37} which is not. In order to see the later, note that M_{37} aligns the pair of paths $p = (\varepsilon, \varepsilon)$, while $out_{p^{-1}\tau_{flip}}(\varepsilon) = out_{\tau_{flip}}(\varepsilon) = \mathsf{P}(\bot, \bot)$. This shows that M_{37} does not output* P *at the root as soon as possible. For similar reasons, the* DTopI *$N_{43} = (M_{43}, dom(\tau_{flip}))$ is not earliest. The* DTop *M_{43}, however, is earliest, since the range of $[\![M_{43}]\!]$ contains #, so that any early output of $\mathsf{P}(\ldots, \ldots)$ would not be correct.*

Note that without domain inspection, earliest DTops are less expressive than DTops in general. The next example shows that domain inspection is needed in order to make some DTops earliest.

Example 67. *The identity function with domain $\{f(c, a),\ f(c, b)\}$ can be computed by some* DTop*. However, if we want this* DTop *to be earliest, then its axiom must produce $f(c, \bot)$ right away. It remains to represent the residual at the pair $(\varepsilon, f2)$, which is the partial function $\{(f(c, a), a), (f(c, b), b)\}$. This residual can be recognized by some* DTopI*, as shown in Example 40, but not by any* DTop *without inspection. Therefore, the above partial identify function is not definable by any earliest* DTop *without inspection, i.e., domain inspection may be required for making* DTop*s earliest.*

Our aim is to restate Theorem 11 of [Engelfriet et al., 2009], that shows that every DTopI$_{\text{td}}$ is equivalent to some earliest DTopI$_{\text{td}}$[1], but also extend it to the more general DTopI$_{\text{reg}}$ case.

In order to do so, we start with a lemma that shows that the axioms and transition rules of earliest DTopIs have a specific form depending on the largest common outputs of the transformation and its residuals at syntactically aligned paths.

Lemma 68. *Let $N = (M, D)$ be an earliest* DTopI*, with $\tau = [\![N]\!]$, and $M = (Q, F, G, \{ax\}, rhs)$. Then:*

(1) if $\tau \neq \emptyset$, $out_\tau(\varepsilon) = ax[q\langle x_0\rangle/\bot \mid q \in Q]$

(2) for every (q, f) such that $rhs(q, f)$ is defined and any pair p syntactically aligned in q, we have $out_{p^{-1}\tau}(f) = rhs(q, f)[q\langle x_i\rangle/\bot \mid q \in Q, x_i \in X]$

Proof. The proof basically relies on the definitions of syntactic alignment and earliestness, and Proposition 49. It should be noticed that both statements would go wrong without assuming trimmedness. Note that useless rules in M may have any form without preventing N from being earliest.

(1) For any $s \in dom(\tau)$, we have $\tau(s) = ax[q\langle x_0\rangle/[\![M]\!]_q(s) \mid q \in Q]$. This means that for all v such that $ax \models v$, for all $s \in dom(\tau)$, $\tau(s) \models v$. Hence, $out_\tau(\varepsilon) \models v$. This would also be true for a npath vf, for $f \in F$. For v such that $v^{-1}ax = q\langle x_0\rangle$, then $\varepsilon \sim_q v$. Since N is earliest, $out_{(\varepsilon,v)^{-1}\tau}(\varepsilon) = \bot$. Hence, $v^{-1}out_\tau(\varepsilon) = \bot$.

(2) Let $p = (u, v)$ syntactically aligned in q, and a tree $s \in dom(\tau)$ such that $s \models u$. Proposition 49 gives that $v^{-1}\tau(s) = [\![M]\!]_q(u^{-1}s)$. If $s \models ufi$, i.e. $u^{-1}s = f(s_1 \dots s_k)$, then by definition of $[\![M]\!]_q$,
$[\![M]\!]_q(f(s_1, ..., s_n)) = rhs(q, f)[q'\langle x_i\rangle/[\![M]\!]_{q'}(s_i) \mid q' \in Q, x_i \in X]$.
This means that if for all paths v' such that $rhs(q, f) \models v'$, for all $s' = f(s_1, ..., s_n) \in u^{-1}dom([\![M]\!]_q)$, then $[\![M]\!]_q(s') \models v'$. From Lemma 55, we know that for all $s' \in dom(p^{-1}\tau)$, $p^{-1}\tau(s') = [\![M]\!]_q(s)$. Hence, $out_{p^{-1}\tau}(f) \models v'$. This would also be true for a npath $v'f$, for $f \in F$.
Furthermore, if $v'^{-1}rhs(q, f) = q'\langle x_i\rangle$, then (ufi, vv') is syntactically aligned in state q'. Since N is earliest, $out_{(ufi,vv')^{-1}\tau}(\varepsilon) = \bot$. That is to say, $v'^{-1}out_{(u,v)^{-1}\tau}(fi) = \bot$. Hence, if $v'^{-1}rhs(q, f) = q'\langle x_i\rangle$, $v'^{-1}out_{p^{-1}\tau}(f) = \bot$.

□

[1]This may take doubly exponential time in the worst case, but only quadratic time if the given transducer is total.

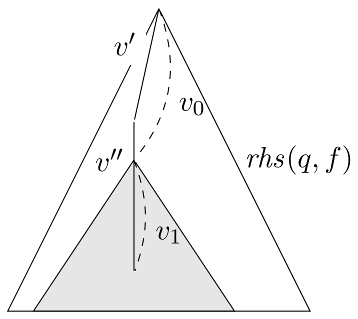


Figure 4.3: Updating the advance of a state $[q, v']$ after reading f

We next show that any DTopI$_{\text{reg}}$ can be made in an equivalent earliest trimmed compatible DTopI$_{\text{reg}}$. This result is an extension of what can be found in [Engelfriet et al., 2009], which demonstrated a similar result for the particular case of DTopI$_{\text{td}}$.

Proposition 69. *Every* DTopI$_{reg}$ $N = (M, D)$ *is equivalent to some compatible earliest* DTopI$_{reg}$ $N' = (M', D')$. *Every* DTopI$_{td}$ $N = (M, D)$ *is equivalent to some compatible earliest* DTopI$_{td}$ $N' = (M', D')$.

Proof. Let $N = (M, D)$ where $\tau = [\![N]\!]$ and $M = (Q, F, G, \{ax\}, rhs)$. Corollary 63 tells us that we can suppose w.l.o.g that N is a trimmed compatible transducer. For any state q of N we define the transformation $[\![N]\!]_q$ by $[\![N]\!]_q = [\![M]\!]_{q|u^{-1}dom([\![N]\!])}$ where (u, v) is some pair aligned in q. Such a pair p exists for all q since N is trimmed. Which pair p aligned in q is chosen does not matter since N is compatible.

We prove both cases by the same construction of $N' = (Q', F, G, \{ax'\}, rhs')$ from N. We define the inspection domain of N' by $D' = dom(\tau)$. Proposition 44 shows that, (1) D' is regular if D was, and (2) that D' is Dtta-recognizable if D was. In order to prove the proposition, it is thus sufficient to construct a DTop M' such that $N' = (M', D')$ is compatible, earliest, and $[\![N']\!] = \tau$.

The idea behind the construction of M' is to produce states of M that produces their output "in advance". If state q of M is not earliest (i.e. if $out_{[\![N]\!]_q}(\varepsilon)$ is not $\perp$), we want to create states $[q, v']$ where $v^{-1}out_{[\![N]\!]_q}(\varepsilon) = \perp$, such that if (u, v) are aligned in q for N, (u, vv') are aligned in $[q, v']$ in N'.

Since N' must be earliest, the axiom and rules of M' must have a special form as stated by Lemma 68. In particular,

$$ax' = out_{[\![N]\!]}(\varepsilon)\Phi$$

for some substitution Φ that maps F-paths leading to $\perp$-leafs to $Q' \times \{x_0\}$. To know how to replace a $\perp$-leaf under path v, we say that if $v^{-1}out_\tau(\varepsilon) = \perp$, then there exists v_0, v_1 such that $v = v_0v_1$, and $v_0^{-1}ax = q\langle x_0\rangle$ for some state $q \in Q$. It is easy to see that $v_1^{-1}[\![N]\!]_q = (v_0v_1)^{-1}\tau = \perp$. Then in M', we choose $v^{-1}ax' = [q, v_1]\langle x_0\rangle$.

Motivated by Lemma 68, if $rhs(q, f)$ exists, we define the rule $rhs'([q, v'], f)$ of M' as follows:

$$rhs'([q, v'], f) = v'^{-1}out_{[\![N]\!]_q}(f)\Phi$$

for some substitution Φ that maps F-paths leading to $\perp$-leafs to $Q' \times X$. For a path v'' such that $(v'v'')^{-1}out_{[\![N]\!]_q}(f) = \perp$, we want to know what state to call under v'' in $rhs'([q, v'], f)$. We say that in $rhs(q, f)$, there is a path v_0 such that v_0 is a prefix of $v'v''$, and $v_0^{-1}rhs(q, f) = q'\langle x_i\rangle$ (see Figure 4.6). When the earliest transducer M' produces v'' on top of its advance of v', the original transducer M only produces v_0 in rule $rhs(q, f)$. This leaves an advance of v_1, which means $v_1^{-1}out_{[\![N]\!]_{q'}}(\varepsilon) = \perp$. Therefore, the new advance is $v_1 = v_0^{-1}v'v''$. We then have $v''^{-1}rhs'([q, v'], f) = [q', v_1]\langle x_i\rangle$.

For the correctness of the construction, we prove that we indeed constructed a transducer that produces its output "ahead" of N. We will prove by induction that $[\![N']\!]_{[q,v']} = v'^{-1}[\![N]\!]_q$.

For a tree $s = f(s_1, ..., s_n)$, we will show that for all output paths v''', if $[\![N']\!]_{[q,v']}(s) \models v'''$, then $[\![N]\!]_q(s) \models v'v'''$. We differentiate two cases.

If $rhs'([q, v'], f) \models v'''$, then $v'^{-1}out_{[\![N]\!]_q}(f) \models v'''$, which means $out_{[\![N]\!]_q}(f) \models v'v'''$. By definition of out, this implies $[\![N]\!]_q(s) \models v'v'''$.

If $rhs'([q, v'], f) \not\models v'''$, there are paths $v'', v_{[q',v_1]}$ such that $v''' = v''v_{[q',v_1]}$ and $v''^{-1}rhs'([q, v'], f) = [q', v_1]\langle x_i\rangle$. This means that $[\![N']\!]_{[q',v_1]}(s_i) \models v_{[q',v_1]}$. We use the same notations as above and in Figure 4.6: there exists v_0 a prefix of $v'v''$, such that $v_0^{-1}rhs(q, f) = q'\langle x_i\rangle$, and $v_0v_1 = v'v''$. By induction hypothesis, we have $[\![N']\!]_{[q',v_1]}(s_i) = v_1^{-1}[\![N]\!]_{q'}(s_i)$. Since $[\![N']\!]_{[q',v_1]}(s_i) \models v_{[q',v_1]}$, we have $[\![N]\!]_{q'}(s_i) \models v_1v_{[q',v_1]}$. Since $v_0^{-1}rhs(q, f) = q'\langle x_i\rangle$, and from the definition of $[\![N]\!]_q$, we have $[\![N]\!]_q(s) \models v_0v_1v_{[q',v_1]}$. Since $v_0v_1 = v'v''$ and $v''' = v''v_{[q',v_1]}$, we have $[\![N]\!]_q(s) \models v'v''v_{[q',v_1]}$, and thus $[\![N]\!]_q(s)v'v'''$.

Note that this also proves that N' is earliest: for the state $[q, v']$, we have that $v'^{-1}out_{[\![N]\!]_q}(\varepsilon) = \perp$. Since $[\![N]\!]'_{[q,v']} = v'^{-1}[\![N]\!]_q$, we have that $out_{[\![N']\!]_{[q,v']}}(\varepsilon) = v'^{-1}out_{[\![N]\!]_q}(\varepsilon) = \perp$. This is true for all states of M'. This means that N' is compatible: if two pairs p, p' are syntactically aligned in $[q, v']$ have the same residual $v'^{-1}[\![N]\!]_q$. Furthermore, N' is earliest: if a pair p is syntactically aligned in $[q, v']$, its residual is $p^{-1}\tau = v'^{-1}[\![N]\!]_q$. Since $v'^{-1}out_{[\![N]\!]_q}(\varepsilon) = \perp$, we have $out_{p^{-1}\tau}(\varepsilon) = \perp$. □

In contrast to Proposition 69, there exists DTopI with inspection by path-closed domains, that cannot be made earliest. Indeed, if the domain is path-closed but not regular, the finiteness statement from Corollary 57 may not hold. This can be seen in the following counter-example.

Example 70. *We consider the partial identity function with the path-closed nonregular domain $D = \{a(a(a(...(\#)))) \mid 2^n$ symbols $a, n \geq 0\}$. This partial function is definable by some* DTopI (M, D) *where $[\![M]\!]$ is the total identity function. However, it cannot be defined by any earliest* DTopI *with inspection by some path-closed domain. Indeed, suppose that such an earliest transducer reads the $2^k + 1$'th symbols a for some k. It then has to produce 2^k symbols a at once. But no* DTop *can do this for all k, since it would need a different state for all k, of which there are infinitely many.*

Note that earliest and compatibility are not enough to obtain a normal form on DTopI$_{\text{reg}}$: the earliest transducer constructed in the proof of Proposition 69 depends heavily on initial choice of a DTopI defining the transformation.

Example 71. *We consider the partial function which maps maps $f(a, a)$ to a and $f(b, b)$ to b. This partial function can be defined by two different earliest* DTopI$_{reg}$ *by domain $D = \{f(a, a), f(b, b)\}$, which is not path-closed so that it is not definable by any* Dtta*. The first* DTop *outputs the subtree at path $f1$ and the second* DTop *outputs the subtree at path $f2$. For the first transducer, the pair $(f1, \varepsilon)$ is syntactically aligned but not the pair $(f2, \varepsilon)$, while it is the converse for the second transducer.*

This example shows that the same transformation can be defined by two different earliest compatible DTopIs with the same regular inspection domain, so that the same output is produced from two different input paths. In this case, the syntactically aligned pairs differ for these two earliest DTopIs. As we will see later on, this problem cannot appear for earliest DTopI$_{\text{td}}$. For this reason, the normal form and learning algorithm that we will develop are restricted to the class DTopI$_{\text{td}}$.

4.7 Semantic Equivalence

We introduce a semantic notion of aligned paths that applies to transformations rather than transducers. The intuition of this semantic alignment is that a pair is semantically aligned if it is susceptible to be a syntactically aligned pair in an earliest transducer. This leads us to a semantic equivalence relation $\equiv_{[\![N]\!]}$ which depends only on the transformation and not on the transducer.

We will show for any DTopI$_{\text{td}}$ that semantic and syntactic alignments are identical. This will leads us to a Myhill-Nerode type Theorem in the more restricted case of top-down inspection.

4.7.1 Semantic Alignments

We next introduce a notion of semantically aligned pairs. We make this notion to identify potential candidates for being syntactically aligned pairs in an earliest DTopI. In essence, if $p = (u, v)$ is to be a syntactically aligned pair, it should at have a functional residual, and for it to be a syntactically aligned pair in an earliest transducer, it should additionally verify that v is as much of the output as one can guess from reading u in the input.

Definition 72. *A pair $p = (u, v)$ is said to be* (semantically) aligned for a partial function τ *if the residual $p^{-1}\tau$ is a partial function, and $v^{-1}out_\tau(u) = \perp$.*

We now prove a useful equivalence, to define semantically aligned pairs in another equivalent way.

Lemma 73. *For any pair $p = (u, v)$ and transformation τ:*

$$v^{-1}out_\tau(u) = \perp \textit{ if and only if } out_\tau(u) \models v \textit{ and } out_{p^{-1}\tau}(\varepsilon) = \perp.$$

Proof. We first prove the implication from the left to the right. For this we assume $v^{-1}out_\tau(u) = \perp$. Then clearly, $out_\tau(u) \models v$. Furthermore, there must exist two trees $s_1, s_2 \in dom(\tau)$ such that $s_1 \models u$, $s_1 \models u$, and $v^{-1}\tau(s_1) \sqcap v^{-1}\tau(s_2) = \perp$. By definition, $p^{-1}\tau$ contains the pairs $(u^{-1}s_1, v^{-1}\tau(s_1))$ and $(u^{-1}s_2, v^{-1}\tau(s_2))$. This means that $out_{p^{-1}\tau}(\varepsilon) \leqslant v^{-1}\tau(s_1) \sqcap v^{-1}\tau(s_2)$, and thus $out_{p^{-1}\tau}(\varepsilon) = \perp$.

We next prove the inverse implication. Let us assume $out_\tau(u) \models v$ and $out_{p^{-1}\tau}(\varepsilon) = \perp$. There must exist two trees $s'_1, s'_2 \in u^{-1}dom(\tau)$, such that $(p^{-1}\tau)(s'_1) \sqcap (p^{-1}\tau)(s'_2) = \perp$. By definition, this means that there exists two trees $s_1, s_2 \in dom(\tau)$ such that $u^{-1}s_1 = s'_1$, $u^{-1}s_2 = s'_2$, and $v^{-1}\tau(s_1) = (p^{-1}\tau)(s'_1)$, $v^{-1}\tau(s_2) = (p^{-1}\tau)(s'_2)$. This means that $\tau(s_1) \sqcap \tau(s_2) \models v\perp$. Hence, $out_\tau(u) \leqslant v\perp$. However, since we assumed $out_\tau(u) \models v$, we have $out_\tau(u) \models v\perp$, and thus, $v^{-1}out_\tau(u) = \perp$. □

While the definitions of $v^{-1}out_\tau(u)$ and $out_{p^{-1}\tau}(\varepsilon)$ seem similar, they are not equivalent without the supposition that $out_\tau(u) \models v$. The following example show that the inverse of the Lemma 73 would not hold without assuming so.

Example 74. *Let $F = G$ and τ be the identity transformation on $\mathcal{T}_F$, i.e. $\tau(s) = s$ for all $s \in \mathcal{T}_F$. All pair (u, u) are semantically aligned and have same residual which is τ. We next consider pairs $p = (u, v)$ where $u = \varepsilon$ and $v \neq \varepsilon$. First note that $out_\tau(u) = out_\tau(\varepsilon) = \perp$. Hence, $v^{-1}out_\tau(u)$ is undefined since we assumed $v \neq \varepsilon$. However, $out_\tau(u) = \perp$, so that $out_\tau \not\models v$, i.e. p is not semantically aligned. Nevertheless, the residual $p^{-1}\tau$ is the partial function which maps all the trees $s \in \mathcal{T}_F$ that satisfy $s \models v$ to their subtree $v^{-1}s$. The image of this partial function is the set $\mathcal{T}_F$, so that $out_{p^{-1}\tau}(\varepsilon) = \perp$. This shows that the inverse of the Lemma 73 would not hold without assuming $out_\tau(u) = \perp$.*

Note that Lemma 73 implies for all pairs p semantically aligned for τ that their residual is nonempty, since $out_{p^{-1}\tau}(\varepsilon) = \bot$.

Lemma 75. *For any earliest* DTopI N*, any two paths that are syntactically aligned in some state of* N *are semantically aligned for* $[\![N]\!]$.

Proof. Let p be a pair of paths that are syntactically aligned in some state of N. Since p is syntactically aligned, Lemma 55 shows that $p^{-1}[\![N]\!]$ is a partial function, and that $out[\![N]\!](u) \models v$. Thus Lemma 73 yields $v^{-1}out_{[\![N]\!]}(u) = \bot$. Furthermore, Since N is earliest, it follows that $out_{p^{-1}[\![N]\!]}(\varepsilon) = \bot$. Hence, p is semantically aligned for $[\![N]\!]$. □

In the case of $\text{DTopI}_{\text{reg}}$ in general, however, not all semantically aligned pairs of a transformation τ are realized into syntactically aligned pairs by an earliest DTopI computing τ.

Example 76. *We reconsider the partial function from Example 71 which is* $[f(a,a)/a,\ f(b,b)/b]$*. It semantically aligns the pairs* $(\varepsilon, \varepsilon)$*,* $(f1, \varepsilon)$*, and* $(f2, \varepsilon)$*. Two earliest* DTopI*s with regular inspection defining this partial function were given in Example 71. The first produces the output at the input path* $f1$ *so the paths* $(\varepsilon, \varepsilon)$ *and* $(f1, \varepsilon)$ *are syntactically aligned, but not* $(f2, \varepsilon)$*. The second* DTopI *produces its output at path* $f2$*. It aligns* $(\varepsilon, \varepsilon)$ *and* $(f2, \varepsilon)$ *syntactically, but not* $(f1, \varepsilon)$*. This shows that not all semantic alignments need to be realized syntactically by all* DTopI*s with regular inspection.*

4.7.2 Semantic Equivalence

For any transformation τ, we now define an equivalence relation $\equiv_\tau$ between pairs p_1 and p_2 of paths that are semantically aligned by τ, as follows:

$$p_1 \equiv_\tau p_2 \text{ iff } p_1^{-1}\tau = p_2^{-1}\tau.$$

Lemma 77. *For* N *an earliest* DTopI *and* p_1, p_2 *two pairs syntactically aligned in some state of* N*, syntactic equivalence* $p_1 \equiv_N p_2$ *implies semantic equivalence* $p_1 \equiv_{[\![N]\!]} p_2$.

Proof. If $p_1 \equiv_N p_2$ then p_1 and p_2 are syntactically aligned, so they are also semantically aligned by Lemma 75, since N is earliest. Furthermore, syntactic equivalence requires $p_1^{-1}[\![N]\!] = p_2^{-1}[\![N]\!]$, so that semantic equivalence follows. □

We now endeavour to obtain a Myhill-Nerode type Theorem for the DTopIs with top-down inspection, that is for the class of DTopI_{td}. **Most of the results that will follow would fail for more general regular inspection**.

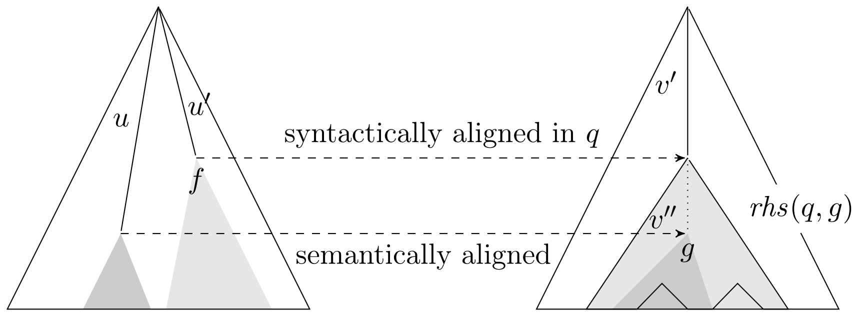


Figure 4.4: Input path u' that produces the node at output path $v = v'v''$.

We wish to prove that the semantic equivalence $\equiv_{[\![N]\!]}$ has finite index, and know from Corollary 57 that the syntactic equivalence $\equiv_N$ has finite index. Therefore, we will show that the classes of syntactic and semantic equivalence classes coincide for any earliest $\mathrm{DT}\textsc{op}\mathrm{I}_{\mathrm{td}}$.

Theorem 78. *For any earliest* $\mathrm{DT}\textsc{op}\mathrm{I}_{td}$ N*, every semantically aligned pair* p *is syntactically aligned.*

Proof. Suppose that $p = (u, v)$ is a semantically aligned pair that is not syntactically aligned. We now consider a tree s such that $s \models u$. We consider the syntactic alignment that produce the node under v when the transducer N computes $[\![N]\!](s)$, as described by Proposition 52. This is the alignment $p' = (u', v')$ where the node under v is not produced yet, but will be after reading the node under u' in s. Formally, this means that p' is syntactically aligned in state q, such that $s \models u'f$ for some $f \in F$, $[\![N]\!](s) \models v'g$ for some $g \in G$, v' is a prefix of v such that $v = v'v''$, and $rhs(q, f) \models v''g$.

Since N is earliest, and p' is syntactically aligned, it follows that p' is semantically aligned by Lemma 75, and thus we have that $v'^{-1} out_\tau(u') = \bot$. Since p is semantically aligned, we have that $v^{-1} out_\tau(u) = \bot$. We will prove this situation to be impossible by distinguishing 4 cases: $u = u'$, u is a prefix of u', u' is a prefix of u, or the last possible case: u and u' are disjoint.

u **equals** u'**.** Assume that $u' = u$. We have $v^{-1} out_\tau(u) = \bot$, and $v'^{-1} out_\tau(u) = \bot$. Since v' is a prefix of v, this implies that $v' = v$. Hence, $p = p'$. This means that p is syntactically aligned, which is in contradiction with our assumption.

u **is a prefix of** u'**.** Suppose that u is a strict prefix of u'. Then by the recursive definition of syntactic alignments, there exists a syntactically aligned pair (u, v'') for some prefix v'' of v', and therefore, of v. Since we supposed N earliest, (u, v'') is semantically aligned, which means $v''^{-1} out_\tau(u) = \bot$. Since (u, v) is semantically aligned, we also have $v^{-1} out_\tau(u) = \bot$. This means that $v'' = v$, and thus that p is syntactically aligned, which is in contradiction with our assumption.

u' **is a prefix of** u**.** Suppose that u' is a strict prefix of u. Hence $u'f$ is a prefix of u too. From the assumption that $rhs(q, f) \models v''g$, we have that $out_\tau(u'f) \models v'v''g$. Since $u'f$ is a prefix of u, we have $out_\tau(u) \models vg$. This means that $v^{-1}out_\tau(u) \neq \bot$, which is in contradiction with the assumption that p is semantically aligned.

u **and** u' **are disjoint.** This case leads to a contradiction to the path-closedness of the domain. Since $dom([\![N]\!])$ is recognized by a DTTA, it is path-closed. This means that we can change s by replacing $u^{-1}s$ by any tree $s' \in u^{-1}dom([\![N]\!])$ while staying in the domain. However, since $p'^{-1}[\![N]\!]$ is functional, and $u'^{-1}s$ did not change, we have that $v'^{-1}[\![N]\!](s[u/s']) = v'^{-1}[\![N]\!](s)$. Notably, $v^{-1}[\![N]\!](s[u/s']) = v^{-1}[\![N]\!](s)$. Hence, $p^{-1}[\![N]\!](u^{-1}s) = p^{-1}[\![N]\!](s')$. Since this is true for all $s' \in u^{-1}dom([\![N]\!])$, we have that $p^{-1}[\![N]\!]$ is constant. However, since p is a semantically aligned, this is a contradiction, as it would prevent $out_{p^{-1}[\![N]\!]}(\varepsilon) = \bot$.

Since all cases are impossible, our assumption is impossible. Hence, there is no semantic aligned pair p that is not syntactically aligned. □

This theorem leads us directly to our desired Myhill-Nerode type Theorem for semantically aligned pairs:

Corollary 79. *For any* DTopI$_{td}$ *N the semantic equivalence relation* $\equiv_{[\![N]\!]}$ *has finite number of equivalence classes.*

Proof. This is an immediate consequence of Theorem 78 on the coincidence of syntactic and semantic alignments for earliest DTopI$_{\text{td}}$, and the fact that the number of equivalence classes for syntactic aligned pairs is finite, as stated in Corollary 57. □

4.7.3 Semantic Successors

In analogy to Definition 47 of syntactic successors, we now define a notion of semantic successors. However, as shown in Example 76, there may be several semantic successors for a single triple (p, f, v'):

Definition 80. *Let* $N = (M, D)$ *be a* DTopI *and* $\tau = [\![N]\!]$*. For any semantically aligned pair* $p = (u, v)$ *of* τ*, input symbol* $f \in F$ *and output path* v' *with* $v'^{-1}out_{p^{-1}\tau}(f) = \bot$*, we define the sets of semantic indexes and semantic successors as follows:*

- $Ind_\tau(p, f, v') = \{i \mid (ufi, vv') \text{ semantically aligned}\}$,

- $Succ_\tau(p, f, v') = \{(ufi, vv') \mid i \in Ind_\tau(p, f, v')\}$.

Furthermore, we define $Succ_\tau(p)$ as the set of all semantic successors of p with respect to τ and some f and v', that is:

$$Succ_\tau(p) = \{p' \in Succ_\tau(p, f, v') \mid f \in F,\ v'^{-1}out_{p^{-1}\tau}(f) = \bot\}$$

Proposition 78 give valuable informations on semantically aligned pairs: since they are exactly the syntactically aligned pairs of an equivalent earliest compatible transducer, all the properties of syntactically aligned pairs can be lifted to semantically aligned pair. The following corollary shows a useful property that transducers with top-down inspection share, that allows for a characterization of their normal form, and later, their learning algorithm.

Corollary 81. *For any earliest* $\mathrm{DT}\textsc{op}\mathrm{I}_{td}$ N*, if* (ufi, v) *is a semantically aligned pair of* $[\![N]\!]$*, then there is no index* j *different from* i *such that* (ufj, v) *is a semantically aligned pair of* $[\![N]\!]$*.*

Proof. Let ufi and ufj be disjoint input paths. If both (ufi, v) and (ufj, v) were semantically aligned, then for all s such that $s \models uf$, $v^{-1}[\![N]\!](s)$ depends functionally of both $ufi^{-1}s$ and $ufj^{-1}s$. As seen in the proof of Proposition 78, this leads to a contradiction. □

We can thus equate syntactic and semantic successors.

Lemma 82. *For any earliest* $\mathrm{DT}\textsc{op}\mathrm{I}_{td}$ N *defining* $\tau = [\![N]\!]$*, any semantically aligned pair* $p = (u, v)$ *of* τ*, input symbol* $f \in F$*, and output path* v' *such that* $v'^{-1}out_{p^{-1}\tau}(f) = \bot$*.*

- $Ind_\tau(p, f, v') = \{ind_N(p, f, v')\}$
- $Succ_\tau(p, f, v') = \{succ_N(p, f, v')\}$

Proof. Since p is semantically aligned, Proposition 78 ensures that it is also syntactically aligned in some state q of N. Lemma 68 then gives us that $out_{p^{-1}[\![N]\!]}(f) = rhs(q, f)[q\langle x_i\rangle/\bot \mid q \in Q, x_i \in X]$. This means that $v'^{-1}out_{p^{-1}\tau}(f) = \bot$ if and only it $v'^{-1}rhs(q, f) = q'\langle x_i\rangle$ for some state q'. This in turns means that $(ufi, vv') \in Succ_\tau(p, f, v')$ if and only if $(ufi, vv') = succ_N(p, f, v')$. Since Corollary 81 indicates that i is the only index such that $(ufi, vv') \in Succ_\tau(p, f, v')$, we also get that $i \in Ind_\tau(p, f, v')$ if and only if $i = ind_N(p, f, v')$. □

In the case of top-down inspection, we thus have *unique* semantic indexes and successors that we can denote by $ind_\tau(p, f, v')$ and $succ_\tau(p, f, v')$.

Another important consequence of top-down inspection is that $Succ_\tau$ can be obtained from $Succ_N$, so that $Succ_\tau$ inherits the inductive nature of $Succ_N$.

Lemma 83. *Let* τ *be a* $\mathrm{DT}\textsc{op}\mathrm{I}_{td}$ *transformation,* $p = (u, v)$ *be a semantically aligned pair of* τ*. Then either* $u = \varepsilon$*, or there exists a semantically aligned pair* p' *of* τ *such that* $p \in Succ_\tau(p')$*.*

Proof. Let N an earliest DTopI$_{\text{td}}$ such that $[\![N]\!] = \tau$. Proposition 78 implies that since p is a semantically aligned pair, then there is a state q in N such that $u \sim_q v$. The recursive nature of syntactically aligned pairs presented in Definition 45 implies that if $u = u'fi$, then there exists v', v'' such that $v = v'v''$, $u' \sim_{q'} v'$, and $v''^{-1}rhs(q', f) = q\langle x_i \rangle$. By applying Proposition 78 again we get that p' is semantically aligned, and p is a successor of p'. □

The combination of Proposition 78 and 52 has an interesting consequence for DTopI$_{\text{td}}$, on which formulation of the learning algorithm in [Lemay et al., 2010] was based: the image of any output in a tree of $\tau(s)$ can be semantically linked to a unique input path, i.e. any output node has a unique *td-origin* in the terminology of [Lemay et al., 2010].

Proposition 84. *Let τ be a* DTopI$_{td}$ *transformation and $s \in dom(\tau)$ an input tree. Then for any output path vg such that $\tau(s) \models vg$, either $out_\tau(\varepsilon) \models vg$ or there exist a unique decomposition $v = v'v''$, an input path $u'f$, such that:*

- $s \models u'f$,
- (u', v') *semantically aligned pair, and*
- $out_{p'^{-1}\tau}(f) \models v''g$.

Proof. Let N be an earliest DTopI$_{\text{td}}$ such that $[\![N]\!] = \tau$. Proposition 84 ensures that for any output path vg such that $\tau(s) \models vg$, either $ax \models vg$ or there exist a unique decomposition $v = v'v''$, an input path $u'f$, such that:

- $s \models u'f$,
- (u', v') is syntactically aligned in some state q of N, and
- $rhs(q, f) \models v''g$.

Since N is earliest, Lemma 68 ensures that $out_\tau(\varepsilon) = ax$. This means that $out_\tau(\varepsilon) \models vg$ if and only if $ax \models vg$. Otherwise, Proposition 78 ensures that (u', v') is semantically aligned. Finally, since N is earliest, Lemma 68 ensures that $out_{p'^{-1}\tau}(f) \models vg$ if and only if $rhs(q, f) \models vg$.

The uniqueness of such a pair is also ensured by Proposition 78: Suppose that there is another pair $p'' = (u'', v'')$ such that:

- $s \models u'f$,
- (u', v') semantically aligned pair, and
- $out_{p'^{-1}\tau}(f) \models v''g$.

Then by Proposition 78, (u'', v'') is syntactically aligned in some state q'. Since N is earliest, Lemma 68 ensures that $out_{p''^{-1}\tau}(f) \models vg$ if and only if $rhs(q', f) \models vg$. This means that p'' fits the criteria of Proposition 84, which is a contradiction, as we know p' is the unique pair to fit those criteria. □

4.8 Unique Normal Forms

Theorem 79 allows us to define a transducer in normal form for transformations defined by a $\mathrm{DT}{\scriptstyle\mathrm{OP}}\mathrm{I}_{\mathrm{td}}$. Closer scrutiny would yield that this normal form coincides with the one described in [Engelfriet et al., 2009]. From its Myhill-Nerode characterization, the normal form of a transformation τ will prove to be the unique minimal earliest trimmed compatible $\mathrm{DT}{\scriptstyle\mathrm{OP}}\mathrm{I}_{\mathrm{td}}$ to define τ, up to state renaming.

We define the normal $\mathrm{DT}{\scriptstyle\mathrm{OP}}\mathrm{I}_{\mathrm{td}}$ $can(\tau)$ based on the Myhill-Nerode like result of Theorem 79, by ensuring that each class of $\equiv_\tau$ is represented by exactly one state in $can(\tau)$. Note that in this part, we will note $[p]_\tau$ (or $[(u,v)]_\tau$) the class of $p = (u,v)$ in $\equiv_\tau$.

Definition 85. *Let τ a transformation definable by a $\mathrm{DT}{\scriptstyle\mathrm{OP}}\mathrm{I}_{td}$. We define its canonical transducer $can(\tau)$ as the pair $(M, dom(\tau))$, where $M = (Q, F, G, ax, rhs)$ such that:*

- *Q is the set of classes $[p]_\tau$ of τ such that $p^{-1}\tau \neq \emptyset$.*
- *$ax = out_\tau(\varepsilon)[v/[(\varepsilon,v)]_\tau\langle x_0\rangle \mid v^{-1}out_\tau(\varepsilon) = \bot]$.*
- *For its rules: for all $p = (u,v)$ such that $[p]_\tau \in Q$ and $f \in G$ such that $uf^{-1}dom(\tau) \neq \emptyset$,*

$$rhs([p]_\tau, f) = out_{p^{-1}\tau}(f)[v'/[(ufi, vv')]_\tau\langle x_i\rangle \,|v'^{-1}out_{p^{-1}\tau}(f) = \bot \text{ and } i = ind_\tau(p,f,v')]$$

We next show that $can(\tau)$ is the unique minimal earliest compatible trimmed $\mathrm{DT}{\scriptstyle\mathrm{OP}}\mathrm{I}$ that defines τ. To show this, we will prove that $[\![can(\tau)]\!] = \tau$, and that $can(\tau)$ is compatible, trimmed and earliest. Then, to prove its minimality, we will consider another earliest compatible trimmed $\mathrm{DT}{\scriptstyle\mathrm{OP}}\mathrm{I}$ that defines τ, and note that it uses equivalent states. Since $can(\tau)$ has no redundant states (no two states equivalent), this will prove that $can(\tau)$ is the unique minimal earliest compatible trimmed $\mathrm{DT}{\scriptstyle\mathrm{OP}}\mathrm{I}$ that defines τ.

Proposition 86. *For any transformation τ definable by some $\mathrm{DT}{\scriptstyle\mathrm{OP}}\mathrm{I}_{td}$, $can(\tau)$ is a $\mathrm{DT}{\scriptstyle\mathrm{OP}}\mathrm{I}_{td}$ defining τ that is compatible, trimmed, and earliest.*

Proof. Suppose that $\tau = [\![N]\!]$ for some $\mathrm{DT}{\scriptstyle\mathrm{OP}}\mathrm{I}_{\mathrm{reg}}$ $N = (M, D)$. Let $can(\tau) = N' = (M', dom(\tau))$. We will first prove that N' defines τ. Then, we will show it is also compatible, trimmed, and earliest.

The first step is to show that $[\![N']\!]_{[p]_\tau} = p^{-1}\tau$. We prove for the input tree $s = f(s_1, ..., s_n)$ that:

$$[\![N']\!]_{[p]_\tau}(s) = out_{p^{-1}\tau}(f)[v'/[\![N']\!]_{[(ufi,vv')]_\tau}(s_i) \,|v'^{-1}out_{p^{-1}[\![N]\!]}(f) = \bot \text{ and } i = ind_\tau(p,f,v')]$$

This is done by induction on the input tree s. By induction hypothesis, $[\![N']\!]_{[(ufi,vv')]_\tau}(s_i) = (ufi, vv')^{-1}\tau(s_i) = v'^{-1}p^{-1}\tau(s)$. We can then prove that $[\![N']\!]_{[p]_\tau}(s) = p^{-1}\tau(s)$. All paths v'' such that $out_{p^{-1}\tau}(f) \models v''$ are both in $p^{-1}\tau(s)$ and in $[\![N']\!]_{[p]_\tau}(s)$. Plus, for v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \perp$, $v'^{-1}[\![N']\!]_{[p]_\tau}(s) = v'^{-1}p^{-1}\tau(s)$.

We add the axiom on top of these production to show that $\tau = [\![N']\!]$. For a tree $s \in dom(\tau)$ we have that $[\![N']\!](s) = out_\tau(\varepsilon)[v/[\![N']\!]_{[(\varepsilon,v)]_\tau}(s) \mid v^{-1}out_\tau(\varepsilon) = \perp]$. We can then prove $[\![N']\!](s) = \tau(s)$. All paths v' such that $out_\tau(\varepsilon) \models v'$ are both in $\tau(s)$ and in $[\![N']\!](s)$. Plus, for v such that $v^{-1}out_\tau(\varepsilon) = \perp$, $v^{-1}[\![N']\!](s) = v^{-1}\tau(s)$.

To show that $[\![N']\!]$ is compatible and earliest, we prove that if a pair $p = (u, v)$ is syntactically aligned, then $u \sim_{[p]_\tau} v$. This can be proven by induction on the length of u. If $u = \varepsilon$, then $\varepsilon \sim_{[p']_\tau} v$ if and only if $v^{-1}ax = [p']_\tau\langle x_0\rangle$. From Definition 85, this means $v^{-1}out_\tau(\varepsilon) = \perp$, and that $[(\varepsilon, v)]_\tau = [p']_\tau$. If $u = u_0fi$, and $u_0 \sim_{[p_0]_\tau} v_0$, and $u \sim_{[p']_\tau} v$, then there exists v_0, v_1 such that $v_1 rhs'([p_0]_\tau, f) = [p']_\tau\langle x_i\rangle$. By induction, we know that $[(u_0, v_0)]_\tau = [p_0]_\tau$. From Definition 85, $v_1^{-1}rhs'([p_0]_\tau, f) = [p']_\tau\langle x_i\rangle$ means that $v_1^{-1}out_{(u_0,v_0)^{-1}\tau}(\varepsilon) = \perp$, and that $[(u_0fi, v_0v_1)]_\tau = [p']_\tau$.

To show that $[\![N']\!]$ is trimmed, we prove that every state $[p]_\tau$ and every rule $rhs'([p]_\tau, f)$ is useful. If $[p]_\tau$ is a state of Q', then p is a semantically aligned pair of τ. Proposition 78 ensures that p is syntactically aligned in some state of N'. As seen above in this proof, this means that p is syntactically aligned in $[p]_\tau$. Hence, $[p]_\tau$ is useful. Furthermore, since $rhs'([p]_\tau, f)$ exists if and only if $p = (u, v)$ and $uf^{-1}dom(\tau) \neq \emptyset$, every rule is also useful. □

Now that we have shown that all DTopI$_{\text{td}}$ have an equivalent earliest trimmed compatible DTopI$_{\text{td}}$, we will prove that for all DTopI$_{\text{td}}$, $can(\tau)$ is the unique minimal earliest trimmed compatible DTopI$_{\text{td}}$ to define τ, up to state renaming. To this end, we will prove that two equivalent earliest trimmed compatible DTopI$_{\text{td}}$ have the same syntactically aligned pairs, which means they use equivalent states. Then, we will consider clear transducers (without two equivalent states) and prove that the only clear earliest trimmed compatible DTopI$_{\text{td}}$ to define τ is $min(\tau)$, up to states renaming.

We start by proving that two equivalent trimmed earliest compatible DTopI$_{\text{td}}$ have the same syntactically aligned pairs.

Theorem 87. *Let $N = (M, D)$ and $N' = (M', D)$ be two equivalent trimmed earliest compatible* DTopI$_{td}$. *For all p syntactically aligned pair of N, p is a syntactically aligned pair of N'.*

Proof. Let $M = (Q, F, G, ax, rhs)$ and $M' = (Q', F, G, ax', rhs')$. This proof is made by induction on the size of u. If $u = \varepsilon$ then for some state q of N, $ax \models vq\langle x_0\rangle$. Since M is earliest, Lemma 68 gives that $v^{-1}out_{[\![N]\!]}(\varepsilon) = \perp$.

For the same reason, since $v^{-1}out_{[\![N']\!]}(\varepsilon) = \bot$, we have that for some q' of N', $ax' \models vq'\langle x_0\rangle$. Hence (u, v) if a syntactically aligned pair of N'.

If $u = u'fi$, then there exists v', v'' such that $v = v'v''$, $u' \sim_{q_0} v'$, and for some state q of N, $rhs(q_0, f) \models vq\langle x_i\rangle$. By induction, there exists a state q_0' of N' such that $u' \sim_{q_0} v'$. This means that $[\![N]\!]_{q_0} = [\![N']\!]_{q_0'}$, and since N and N' are both trimmed and compatible, if $rhs(q_0, f)$ is defined, then $rhs'(q_0', f)$ is defined. Since M is earliest, Lemma 68 gives that $v''^{-1}out_{[\![N]\!]_{q_0}}(f) = \bot$. For the same reason, since $v''^{-1}out_{[\![N']\!]_{q_0'}}(f) = \bot$, we have that for some q' of N', for some index j, $ax' \models vq'\langle x_j\rangle$. The fact that $i = j$ is due to the fact that both are equal to $ind_\tau((u', v'), f, v'')$. □

This important theorem allows us to ensure that a DTopI$_{\text{td}}$ will have as few states as possible when it has exactly one state per semantic class $[p]_\tau$. To ensure this, we define clear transducers as transducers with no redundant states.

Definition 88. *We say a compatible* DTopI N *is* clear *if it is trimmed, and for* q *and* q' *two distinct states of* N, $[\![N]\!]_q \neq [\![N']\!]_{q'}$.

Note that just like for trimmed DTopI, it is easier to prove the existence of aclear DTopI equivalent to some DTopI N than to actually compute it.

Lemma 89. *For* N *an earliest compatible* DTopI *there exists an equivalent clear earliest compatible* DTopI.

Proof. The existence of a trimmed DTopI has already been argued in Lemma 51: if a state or a rule is useless, it can be deleted without changing the semantics of N, or its earliest compatible nature. Similarily, if there exists two equivalent states q, q' such that $[\![N]\!]_q = [\![N]\!]_{q'}$, then one can delete q' and its rules $rhs(q', f)$, nd replace every occurence of $q'\langle x_i\rangle$ by $q\langle x_i\rangle$ in ax and rhs. Since both states are equivalent, this substitution can be done without changing the semantics of N, or its earliest compatible nature. We can thus delete redundant states until none are left, and end up obtaining a clear earliest compatible DTopI equivalent to N. □

We show that our definition of clear implies a minimal number of states: since two equivalent earliest compatible trimmed DTopI$_{\text{td}}$ have identical syntactically aligned pairs, it is easy to show that they use equivalent states.

Lemma 90. *For* $N = (M, D)$ *a trimmed earliest compatible* DTopI$_{td}$, *and* $N' = (M', D)$ *an equivalent clear earliest compatible* DTopI$_{td}$. *There exists an onto function* ϕ *from the states of* N *to the states of* N' *such that for all* q *state of* N, $[\![N']\!]_{\phi(q)} = [\![N]\!]_q$.

Proof. If q is a state of N, a trimmed DTopI, there exists (u, v) such that $u \sim_q v$. From Theorem 87 we conclude that there exists a state q' of N' such that $u \sim_{q'} v$. For such a q', we would have $[\![N]\!]_q = [\![N']\!]_{q'}$. We note $\phi(q)$ the only state q' of the clear DTopI$_{\mathrm{td}}$ such that $[\![N]\!]_q = [\![N']\!]_{q'}$. This function is onto through a symmetrical reasoning: if q' is a state of N', a trimmed DTopI$_{\mathrm{td}}$, there exists (u, v) such that $u \sim_{q'} v$. From Theorem 87 we conclude that there exists a state q of N such that $u \sim_q v$. For such a q, we would have $[\![N]\!]_q = [\![N']\!]_{q'}$, hence $\phi(q) = q'$. □

In the following theorem, we will prove that there only exists one clear earliest compatible DTopI$_{\mathrm{td}}$, up to state renaming, that is to say that if two DTopI$_{\mathrm{td}}$ N and N' are equivalent, clear, earliest and compatible, then there exists a one-to-one function ϕ from the states of N to the states of N' such that if every occurrence of every state q of N is replaced by its image $\phi(q)$, we obtain exactly N'.

Lemma 91. *If N and N' are two equivalent clear earliest compatible* DTopI$_{td}$*, then $N = N'$, up to state renaming.*

Proof. From Lemma 90, we know that there exists a one-to-one correspondence ϕ between the states of N and the states of N' such that $[\![N]\!]_q = [\![N']\!]_{\phi(q)}$. We now show that this one-to-one correspondence is indeed a state rewriting between N and N'. For that, it just remains to prove that the rules of q and $\phi(q)$ are identical up to state renaming. First of all, since $[\![N]\!]_q = [\![N']\!]_{\phi(q)}$ and both N and N' are trimmed, we know that for all input letter f, there exist a rule $rhs(q, f)$ if and only if there is a tree of root f in $dom([\![N]\!]_q)$, if and only if there is a tree of root f in $dom([\![N']\!]_{\phi(q)})$, if and only if there exist a rule $rhs'(\phi(q), f)$. Furthermore, since both N and N' are earliest, from Lemma 68, we know that $rhs(q, f)\Psi = out_{[\![N]\!]_q}(f)$ and $rhs'(\phi(q), f)\Psi' = out_{[\![N']\!]_{\phi(q)}}(f)$ for some Ψ, Ψ'. Since $[\![N]\!]_q = [\![N']\!]_{\phi(q)}$, all that remains to show is that if $v^{-1}rhs(q, f) = q'\langle x_i\rangle$, then $v^{-1}rhs'(\phi(q), f) = \phi(q')\langle x_i\rangle$. Since $[\![N]\!]_q = [\![N']\!]_{\phi(q)}$, we have that if $v^{-1}rhs(q, f) = q'\langle x_i\rangle$, then $v^{-1}rhs'(\phi(q), f) = q''\langle x_j\rangle$. Since N is trimmed, there exists (u_0, v_0) such that $u_0 \sim_q v_0$ in N (and hence $u_0 \sim_{\phi(q)} v_0$ in N'). This means that for N, $i = ind_{[\![N]\!]}((u_0, v_0), f, v)$, and $u_0 f i \sim_{q'} v_0 v$. Hence for N', $i = ind_{[\![N']\!]}((u_0, v_0), f, v)$, and $u_0 f i \sim_{q''} v_0 v$. This means that if $v^{-1}rhs(q, f) = q'\langle x_i\rangle$, then $v^{-1}rhs'(\phi(q), f) = q''\langle x_j\rangle$, with $q'' = \phi(q')$ and $i = j$. □

This finally proves our normal form theorem:

Theorem 92. *For $N = (M, D)$ a* DTopI$_{reg}$*, there exists a unique equivalent compatible earliest* DTopI$_{td}$ *with a minimal number of states, up to state renaming.*

Proof. The existence of such a DTopI is proven by Lemma 89, its uniqueness by Lemma 91. □

4.9 Learning from Examples

We next show how to learn transducers of the class $\mathrm{DTopI}_{\mathrm{td}}$ for a given input domain. Since DTTAs are themselves learnable [Oncina and Garcia, 1992], this is a reasonable assumption to make.

4.9.1 Learning Model

We fix ranked alphabet F and G. Since we suppose the domain of our transformation to be previously known, we define the class of all DTopI that share the same domain D:

Definition 93. *For any top-down domain $D \subseteq \mathcal{T}_F$, we define the class $\mathrm{DTopI}_{td}(D)$ such that it contains all transformations τ from $\mathcal{T}_F$ to $\mathcal{T}_G$ definable by some DTopI_{td} with $dom(\tau) = D$.*

A *sample* is a finite partial function $S \subseteq \mathcal{T}_F \times \mathcal{T}_G$. A sample S is called *compatible* with D if $dom(S) \subseteq D$. A sample S for a transformation τ is a finite subset of $S \subseteq \tau$.

Definition 94. *We say that the class of $\mathrm{DTopI}_{td}(D)$ is learnable if there are:*
- *an algorithm $learn_D$ defining a partial function that maps samples compatible with D to $\mathrm{DTopI}_{td}(D)$ in normal form, and*
- *a function char that maps $\mathrm{DTopI}_{td}(D)$ N in normal form to samples of transformation $[\![N]\!]$.*

We require for any $\mathrm{DTopI}_{td}(D)$ N and any sample S for $[\![N]\!]$ containing $char(N)$ that $learn_D(S) = N$.

There are several parameters to consider when describing the complexity of learning algorithms:
- Sample complexity describes the number of examples in $char(N)$ as a function of the size of N.
- Time complexity describes the complexity of the learning algorithm $learn_D$ as a function of the size of its input sample.

We say a class is learnable *with polynomial ressources* if the the number of examples in $char(N)$ is polynomial as a function of the size of N, and the learning algorithm $learn_D$ is in polynomial time as a function of the size of its input sample.

Theorem 95. *For any D definable by some DTTA, the class $\mathrm{DTopI}_{td}(D)$ is learnable with polynomial resources.*

Proof. The proof captures the rest of this section. It will follow from Propositions 103 and 109. □

Since the exact domain D of the target $\mathrm{DT{\small OP}I_{td}}(D)$ $N = (M, D')$ is assumed to be known, and since the target transducer will be in normal form, know have that $D = D'$, so that we only have to learn the DTop M.

Finally, we will assume that $D \neq \emptyset$, since the case $D = \emptyset$ is easy to treat.

4.9.2 Characteristic Samples

The purpose a characteristic sample to provide enough information to describe a $\mathrm{DT{\small OP}I_{td}}(D)$ N in normal form, where D is recognized by a Dtta. Since normal forms can be characterized in terms of the finitely many equivalence classes of residuals of $[\![N]\!]$, the objective is to present the required information on finitely many pairs p of paths such that the residuals $p^{-1}[\![N]\!]$ represent all relevant classes.

The first question is by which pair p to represent a residual $p^{-1}[\![N]\!]$. The idea is to choose the least pair p' that defines the same residual as p with respect to the following total order. If $p = (u, v)$ and $p' = (u', v')$ the we define $p < p'$ if and only if:

- if $|u| < |u'|$,
- if $|u| = |u'|$ and $u <_{lex} u'$,
- if $u = u'$ and $|v| < |v'|$, or
- if $u = u'$, and $|v| = |v'|$ and $v <_{lex} v'$.

This order is interesting for two reasons. The first one is that contrary to simple simple lexical order, which can produce infinite sets with no minimals (e.g. the language a^*b if $a <_{lex} b$), this order has a well-defined notion of minimals in sets. Furthermore, it has interesting properties, chief amongst them being stability by composition.

Lemma 96. *If $p = (u, v)$, $p' = (u', v')$ two pairs such that $p <_{lex} p'$, then for every pair (u'', v''), $(uu'', vv'') <_{lex} (u'u'', v'v'')$.*

To properly define the notion of a sample S containing enough information to learn a $\mathrm{DT{\small OP}I_{td}}(D)$ N, we will establish what semantically aligned pairs of τ are of relevance, and what S should teach on them. Our first move is to define minimal pairs and their boundary, i.e. all the semantically aligned pairs a sample should have information about in order to learn N.

Definition 97. *For any transformation τ and p a semantically aligned pair of τ, we define $minp_\tau(p) = min\{p' \mid p' \equiv_\tau p\}$. By extension, we define the set of minimal semantically aligned pairs of τ as:*

$$minp(\tau) = \{p \mid p \text{ least semantically aligned pair for } \tau \text{ with residual } p^{-1}\tau\}$$

Note that for any $\text{DTopI}_{\text{td}}(D)$ N the set of residuals of $[\![N]\!]$ is finite by Theorem 79. This means $minp(\tau)$ is finite too.

To find all minimal pairs, our algorithm will explore aligned pairs starting at the axiom, and continue repeatedly with all new pairs that are detected. Hence, the aligned pairs that will be explored do not only contain those in $minp(\tau)$, but also their successors, as well as all the aligned pairs of the axiom.

Definition 98. *For any transformation τ we define the boundary of the minimal semantically aligned pairs of τ as:*

$$\begin{aligned} minp^+(\tau) =& \{(\varepsilon, v) \mid v^{-1}out_\tau(\varepsilon) = \bot\} \cup \\ & \{p' \in Succ_\tau(minp(\tau))\} \end{aligned}$$

The following lemma shows for transformations τ defined by DTopI_{td}'s, all elements of $minp(\tau)$ are in fact part of the axiom, or successors of of some element of $minp(\tau)$:

Lemma 99. *Let N be a DTopI_{td}, and $[\![N]\!] = \tau$. Then $minp(\tau) \subseteq minp^+(\tau)$*

Proof. We consider $p = (u, v) \in minp(\tau)$. We will prove that it is either of form $(\varepsilon, v) \mid v^{-1}out_\tau(\varepsilon) = \bot$ or a successor of $p' \in minp(\tau)$.

If $u = \varepsilon$, then since all elements of $minp(\tau)$ are semantically aligned, this means that $v^{-1}out_\tau(\varepsilon) = \bot$.

If $u = u'fi$ for some u', f, i, then Lemma 83 shows the existence of a pair $p' = (u', v')$ such that p is a successor of p'. Let v'' such that $v = v'v''$. Suppose $p' = (u', v') \notin minp(\tau)$. Then there exists $p_0 = (u_0, v_0)$ such that $p_0 < p'$ and $p_0 \equiv_\tau p'$. Notably, $(u_0fi, v_0v'') < (u'fi, v'v'')$ and $(u_0fi, v_0v'') \equiv_\tau (u'fi, v'v'')$. This is in contradiction with the fact that $p \in minp(\tau)$. Hence, $p' \in minp(\tau)$. □

The set $minp(\tau)$ can be seen as having one unique representative for each class of $\equiv_\tau$. In this sense, it can be seen as representing the states of $can(\tau)$. As a matter of fact, we will define $repcan(\tau)$ the state renaming of $can(\tau)$ where each state $[p]_\tau$ is represented by the unique pair $p' = minp_\tau(p)$.

Definition 100. *Let N a $\text{DTopI}_{td}(D)$, and $\tau = [\![N]\!]$. We define $repcan(\tau)$ the representative of $can(\tau)$ as the DTop $M = (Q, F, G, ax, rhs)$ where:*

- *$Q = minp(\tau)$*
- *$ax = out_\tau(\varepsilon)[v \leftarrow minp_\tau(\varepsilon, v)\langle x_0\rangle \mid v^{-1}out_\tau(\varepsilon) = \bot]$*
- *For $p = (u, v) \in minp(\tau)$, f such that $uf^{-1}D \neq \emptyset$,*

$$\begin{aligned} rhs(p, f) = out_{p^{-1}\tau}(f)[v' \leftarrow & minp_\tau(ufi, vv')\langle x_i\rangle \\ & \mid v'^{-1}out_{p^{-1}\tau}(f) = \bot,\ i = ind_\tau(p, f, v')] \end{aligned}$$

We remember that Lemma 82 ensures that for $p = (u, v)$ semantically aligned, and f, v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \bot$, $ind_\tau(p, f, v')$ is unique, and $(ufi, vv') = succ_\tau(p, f, v')$. This $repcan(\tau)$ is the actual target of our algorithm.

We now consider our sample S, and establish what kind of information it needs to contain for our learning algorithm to be able to retro-engineer $repcan(\tau)$. To this end we will notably have to be able to identify equivalent pairs for $\equiv_\tau$. In order to show that $p \not\equiv_\tau p'$ by a sample S for τ, we will require that p and p' are in contradiction with respect to S in the following sense:

Definition 101. *Given a sample S for a transformation τ, we say that two semantically aligned pairs p and p' are in contradiction with respect to S and write $p \nparallel_S p'$ if $p^{-1}S \cup p'^{-1}S$ is not functional.*

In this case, S contains a counter example for $p \not\equiv_\tau p'$. We now use this definition to formalize what it means for a sample S to be characteristic for τ.

Definition 102. *Let N be a* DTopI$_{td}(D)$ *and $\tau = [\![N]\!]$. A sample S for τ is called* characteristic *if:*

(1) $out_S(\varepsilon) = out_\tau(\varepsilon)$

(2) for all $p \in minp(\tau)$ and $f \in F$: $out_{p^{-1}S}(f) = out_{p^{-1}\tau}(f)$,

(3) for $p = (u, v) \in minp(\tau)$, $f^{(k)} \in F$, for all $j \in \{1, \dots, k\}$, if $(ufj, vv')^{-1}\tau$ is not functional, then $(ufj, vv')^{-1}S$ is not functional.

(4) for all $p \in minp^+(\tau)$, and all $p' \in minp(\tau)$ such that $p^{-1}D = p'^{-1}D$, and $p \not\equiv_\tau p'$: $p \nparallel_S p'$.

Points (1) and (2) ensure that S contains enough information to build the axiom and rules of our target transducer, as seen in Lemma 68. Point (3) ensures we never explore pairs that are not semantic alignments. Point (4) ensures we are able to tell which aligned pairs are equivalent to which minimal pairs.

Note that if a sample S for τ is characteristic, then any larger sample for τ is characteristic too. It remains to show that for all DTopI$_{\mathrm{td}}(D)$ N, there exists a characteristic sample for $[\![N]\!]$.

Proposition 103. *Let D be definable by a* Dtta*. For any* DTopI$_{td}(D)$ *N, where $[\![N]\!] = \tau$, there exists a characteristic sample for τ with a number of examples polynomial in the number of equivalence classes in $\equiv_\tau$.*

Proof. We will show that a polynomial number of examples is required for each point (1-4) of Definition 102.

For point (1), we want to ensure that $out_S(\varepsilon) = out_\tau(\varepsilon)$. First, we need at least one example in to ensure that $out_S(\varepsilon)$ is defined. Note that such an example always exists, as we supposed $dom(\tau) = D \neq \emptyset$. We fix $s_\varepsilon \in dom(\tau)$ arbitrarily and add $(s_\varepsilon, \tau(s_\varepsilon))$ to S. Then, since $S \subseteq \tau$ will be guaranteed, the only concern is that $out_S(\varepsilon)$ is bigger than $out_\tau(\varepsilon)$. To this end, we will provide one example per equivalence class $[p]_\tau$ of $\equiv_\tau$, where $p = (\varepsilon, v)$ is an aligned pair of τ. For each such class $[p]_\tau$, there exists a tree $s_{[p]_\tau}$ such that $v^{-1}\tau(s_{[p]_\tau}) \sqcap v^{-1}\tau(s_\varepsilon) = \bot$. If $(s_\varepsilon, \tau(s_\varepsilon)) \in S$ and for all equivalence class $[p]_\tau$ of $\equiv_\tau$, where $p = (\varepsilon, v)$ is an aligned pair of τ, $(s_{[p]_\tau}, \tau(s_{[p]_\tau})) \in S$, then $out_{pS}(\varepsilon) = out_\tau(\varepsilon)$.

Point (2) works in a similar fashion. We want to ensure that for all $p = (u, v) \in minp(\tau)$, for all input letter f, $out_{p^{-1}S}(f) = out_{p^{-1}\tau}(f)$. First, we need at least one example such that $s_{p,f} \models uf$ to ensure that $out_{p^{-1}S}(f)$ is defined. We call this example $(s_{p,f}, \tau(s_{p,f}))$. Then, since $S \subseteq \tau$, the only concern is that $out_{p^{-1}S}(f)$ is bigger than $out_{p^{-1}\tau}(f)$. We shall ensure that there exists enough examples in S so that for all v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \bot$, $v'^{-1}out_{p^{-1}S}(f) = \bot$. To this end, we will provide one example per equivalence class $[p']_\tau$ of $\equiv_\tau$, where $p' \in Succ_\tau(p)$. For each such class $[p']_\tau$, there exists a tree s such that $p'^{-1}\tau(s) \sqcap p'^{-1}\tau(v^{-1}s_{p,f}) = \bot$. We then choose a tree $s_{p,f,[p']_\tau}$, such that $u^{-1}s_{p,f,[p']_\tau} = s$. If $(s_{p,f}, \tau(s_{p,f})) \in S$ and for all $v'^{-1}out_{p^{-1}\tau}(f) = \bot$, $p' \in Succ_\tau(p)$, $(s_{p,f,[p']_\tau}, \tau(s_{p,f,[p']_\tau})) \in S$, then $out_{p^{-1}S}(f) = out_{p^{-1}\tau}(f)$.

Point (3) is ensured by providing an explicit counterexample for every pair $p' = (ufj, vv')$ we want to prove is not functional. If $p'^{-1}\tau$ is not functional, then there exists $s_{p'}$, $s'_{p'}$ input trees such that $ufj^{-1}s_{p'} = ufj^{-1}s'_{p'}$ but either $vv'^{-1}\tau(s_{p'}) \neq= vv'^{-1}\tau(s'_{p'})$ or $\tau(s_{p'}) \models vv'$ but $\tau(s'_{p'}) \not\models vv'$. Hence, if S contains $(s_{p'}, \tau(s_{p'}))$ and $(s'_{p'}, \tau(s'_{p'}))$, then $p'^{-1}S$ is not functional. Note that there is a polynomial number of those pairs: for $p = (u, v) \in minp(\tau)$, for f an input letter, all paths v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \bot$ are in $rhs(q, f)$ where q is the state of N that computes $p^{-1}\tau$. Since $j \leq rank(f)$, this leaves a polynomial number of pairs to consider.

Point (4) works in a similar fashion. For $p = (u, v)$, $p' = (u', v')$ two non-equivalent semantic alignments, if $p^{-1}D = p'^{-1}D$, then there exists s an input tree such that $p{-}1\tau(s) \neq p'{-}1\tau(s)$. We take two input trees $s_{p,p'}$, $s'_{p,p'}$ such that $u^{-1}s_{p,p'} = u'^{-1}s'_{p,p'} = s$. Hence, if S contains $(s_{p,p'}, \tau(s_{p,p'}))$ and $(s'_{p,p'}, \tau(s'_{p,p'}))$, then $p'^{-1}S$ is not functional. Note that there is a polynomial number of those cases to consider, since $minp(\tau)$ and $minp^+(\llbracket N \rrbracket)$ are of polynomial size themselves.

By taking all examples needed to ensure points (1-4), we built a characteristic sample for τ in polynomial size. □

Example 104. *For the transduction τ_{flip} of Example 37, there are four residuals. The first is the residual of the minimal pair* (P1, P2)*, the identity on lists of* A*, the second is the residual of the minimal pair* (P2, P1)*, the identity* *on lists of* B*, and the others are the residuals of the two semantic aligned pairs from the earliest axiom,* $(\varepsilon, \mathsf{P1})$ *and* $(\varepsilon, \mathsf{P2})$*.*

$$minp(\tau_{\mathit{flip}}) = \{(\varepsilon, \mathsf{P1}),\ (\varepsilon, \mathsf{P2}),\ (\mathsf{P1}, \mathsf{P2}),\ (\mathsf{P2}, \mathsf{P1})\}$$

The boundary contains the aligned pairs from the axiom, and those directly extending the pairs in $minp(\tau_{\mathit{flip}})$*:*

$$minp^{+}(\tau_{\mathit{flip}}) = minp(\tau_{\mathit{flip}}) \cup \{(\mathsf{P1A1}, \mathsf{P2A1}),\ (\mathsf{P2B1}, \mathsf{P2B1})\}$$

To satisfy point (1), a characteristic sample would need enough information to deduce $out_{\tau_{\mathit{flip}}}(\varepsilon)$*. As seen in Proposition 103, this means that we need a first example* $(s_\varepsilon, \tau_{\mathit{flip}}(s_\varepsilon))$*, and another example for all equivalence classes of aligned pairs of form* (ε, v)*. In* τ_{flip} *there are two,* $[(\varepsilon, \mathsf{P1})]_{\tau_{\mathit{flip}}}$ *and* $[(\varepsilon, \mathsf{P2})]_{\tau_{\mathit{flip}}}$*. We choose* $s_\varepsilon = \mathsf{P}(\#, \#)$*. For* $[(\varepsilon, \mathsf{P1})]_{\tau_{\mathit{flip}}}$*, we choose* $s_{[(\varepsilon,\mathsf{P1})]_{\tau_{\mathit{flip}}}} = \mathsf{P}(\#, \mathsf{B}(\#))$*. For* $[(\varepsilon, \mathsf{P2})]_{\tau_{\mathit{flip}}}$*, we choose* $s_{[(\varepsilon,\mathsf{P2})]_{\tau_{\mathit{flip}}}} = \mathsf{P}(\mathsf{A}(\#), \#)$*.*

To satisfy point (2), a characteristic sample would need enough information to deduce $out_{p^{-1}\tau_{\mathit{flip}}}(f)$ *for all relevant pairs* p *and* f*. As seen in Proposition 103, this means that for all* $p = (u, v) \in minp(\tau_{\mathit{flip}})$*,* f *an input letter, we choose an example* $(s_{p,f}, \tau_{\mathit{flip}}(s_{p,f}))$*.*

Then, for all equivalence classes $[p']_{\tau_{\mathit{flip}}}$ *where* $p' \in Succ_{\tau_{\mathit{flip}}}(p)$ *we pick another example* $s_{p,\mathsf{P},[p']_{\tau_{\mathit{flip}}}}$ *to ensure* $v'^{-1}out_{p^{-1}S}(\mathsf{P}) = \perp$*.*
For $p = (\varepsilon, \mathsf{P1})$*, the only letter that can be read is* f*, which leads to the only successor* $p' = (\mathsf{P2}, \mathsf{P1})$*. We choose* $s_{p,\mathsf{P}} = \mathsf{P}(\#, \#)$*, and* $s_{p,\mathsf{P},[p']_{\tau_{\mathit{flip}}}} = \mathsf{P}(\#, \mathsf{B}(\#))$*.*
For $p = (\varepsilon, \mathsf{P2})$*, the only letter that can be read is* f*, which leads to the only successor* $p' = (\mathsf{P1}, \mathsf{P2})$*. We choose* $s_{p,\mathsf{P}} = \mathsf{P}(\#, \#)$*, and* $s_{p,\mathsf{P},[p']_{\tau_{\mathit{flip}}}} = \mathsf{P}(\mathsf{A}(\#), \#)$*.*
For $p = (\mathsf{P1}, \mathsf{P2})$*, two letters can be read:* A *which leads to the only successor* $p' = (\mathsf{P1A1}, \mathsf{P2A1})$*, and* $\#$*, which leads to no successor. For* A*, we choose* $s_{p,\mathsf{A}} = \mathsf{P}(\mathsf{A}(\#), \#)$*, and* $s_{p,\mathsf{A},[p']_{\tau_{\mathit{flip}}}} = \mathsf{P}(\mathsf{A}(\mathsf{A}(\#)), \#)$*. For* $\#$*, we only need to choose* $s_{p,\#} = \mathsf{P}(\#, \#)$
For $p = (\mathsf{P2}, \mathsf{P1})$*, two letters can be read:* B *which leads to the only successor* $p' = (\mathsf{P2B1}, \mathsf{P1B1})$*, and* $\#$*, which leads to no successor. For* B*, we choose* $s_{p,\mathsf{B}} = \mathsf{P}(\#, \mathsf{B}(\#))$*, and* $s_{p,\mathsf{B},[p']_{\tau_{\mathit{flip}}}} = \mathsf{P}(\#, \mathsf{B}(\mathsf{B}(\#)))$*. For* $\#$*, we only need to choose* $s_{p,\#} = \mathsf{P}(\#, \#)$*.*

To satisfy point (3), a characteristic sample would need enough information to deduce which pairs (ufi, vv') *extending a minimal pair* p *are semantic alignments. As seen in Proposition 103, this means that for all* $p = (u, v) \in minp(\tau_{\mathit{flip}})$*, and an extension* $p' = (ufi, vv')$ *that is not a semantic alignment, we need two examples to ensure* $p'^{-1}S$ *is not functional. In* τ_{flip}*, there are only two such pairs we need to consider,* $(\mathsf{P1}, \mathsf{P1})$ *and* $(\mathsf{P2}, \mathsf{P2})$*. For* $(\mathsf{P1}, \mathsf{P1})$*, we choose the pair of examples* $(\mathsf{P}(\#, \#), \mathsf{P}(\#, \#))$

and $(\mathsf{P}(\#, \mathsf{B}(\#)), \mathsf{P}(\mathsf{B}(\#), \#))$. *For* $(\mathsf{P2}, \mathsf{P2})$, *we choose* $(\mathsf{P}(\#, \#), \mathsf{P}(\#, \#))$ *and* $(\mathsf{P}(\mathsf{A}(\#), \#), \mathsf{P}(\#, \mathsf{A}(\#)))$.

To satisfy point (4), a characteristic sample would need enough information to differentiate pairs $p \in minp(\tau_{flip}) \cup minp^+(\tau_{flip})$ *from their non-equivalent counterpart of* $p' \in minp(\tau_{flip})$ *of same domain. As seen in Proposition 103, this means that for all such pair of alignments* p, p', *we need one example to ensure* $p \not\parallel_S p'$. *In* τ_{flip}, *only* $(\varepsilon, \mathsf{P1})$ *and* $(\varepsilon, \mathsf{P2})$ *are of same domain but not equivalent. We choose the example* $(\mathsf{P}(\#, \mathsf{B}(\#)), \mathsf{P}(\mathsf{B}(\#), \#))$.

Hence, a complete characteristic sample would be the transformation:

$$\begin{array}{lll} S = [& \mathsf{P}(\#,\#)/\mathsf{P}(\#,\#)\,, & \\ & \mathsf{P}(\mathsf{A}(\#),\#)/\mathsf{P}(\#,\mathsf{A}(\#))\,, & \mathsf{P}(\#,\mathsf{B}(\#))/\mathsf{P}(\mathsf{B}(\#),\#)\,, \\ & \mathsf{P}(\mathsf{A}(\mathsf{A}(\#)),\#)/\mathsf{P}(\#,\mathsf{A}(\mathsf{A}(\#)))\,, & \mathsf{P}(\#,\mathsf{B}(\mathsf{B}(\#)))/\mathsf{P}(\mathsf{B}(\mathsf{B}(\#)),\#)\,] \end{array}$$

4.9.3 Learning Algorithm

We describe the algorithm $learn_D$. The goal is to create a minimal leftmost earliest $\text{DT}\textsc{op}\text{I}_{\text{td}}(D)$, which means creating earliest states with no redundancy, their rules, and the axiom. The idea is to try to fold any new aligned pair we find to an existing state. If no equivalent state can be found, we create a new one.

In this algorithm we build a $\text{DT}\textsc{op}\text{I}_{\text{td}}(D)$ $learn_D(S) = (M, D)$, where $M = (Q, F, G, ax, rhs)$. For simplicity's sake, our states will be pair (u, v). Those states will be divided in two disjoint sets: Q_{safe} for pairs that minimally represent an equivalence class, and therefore represent a state in $learn_D(S)$, and Q_{temp}, for pairs that have not yet been examined by the algorithm, and are still susceptible to be equivalent to an existing pair in Q_{safe}. Rules of rhs are only created for states of Q_{safe}, but leaves in these rules or the axiom can temporarily be pairs $p\langle x_i \rangle$, where p is still an "unapproved" pair of Q_{temp}.

From Definition 102, we know that if S is characteristic, $p^{-1}\tau$ is functional if and only if $p^{-1}S$ is functional. Furthermore, $p \equiv_\tau p'$ if and only if $p^{-1}D = p'^{-1}D$ and $\neg p \not\parallel_S p'$. Procedure *integrate-state* describes how, given a $\text{DT}\textsc{op}\text{I}$ N and a sample S, to test if a pair $p \in Q_{temp}$ is equivalent to an existing state in Q_{safe} and, if it is not, how to create a new state and its rules. Note that in these rules, $p_{fi}\langle x_i \rangle$ can appear for pairs that are not yet confirmed to be original states. These pairs are added to Q_{temp}. The creation of the axiom works in a similar manner to the way we create a rule in Procedure *integrate-state*. From there, the full algorithm goes as described in Figure 4.5.

Example 105. *We try to learn a transducer for* τ_{flip}, *with the characteristic sample we found in Example 104.*

Our end goal in this part is to prove the correctness of our algorithm, i.e.

```
// let $F$ and $G$ be ranked signatures
// and $D \subseteq \mathcal{T}_F$ the language of some DTTA

fun $learn_D(S)$ // $S \subseteq D \times \mathcal{T}_G$ finite partial function
  $Q_{temp}$ := $\{(\varepsilon, v) \mid v^{-1}out_S(\varepsilon) = \bot\}$
  $ax$ = $out_S(\varepsilon)$ $[v \leftarrow (\varepsilon, v)\langle x_0\rangle \mid (\varepsilon, v) \in Q_{temp}]$
  $Q_{safe}$ := $\emptyset$
  $rhs$ := $\emptyset$
  $M$ = $(Q_{safe} \cup Q_{temp}, F, G, ax, rhs)$
  proc $integrate\text{-}state(p)$ = // fusion temporary state $p$ with
      // some safe state if possible or make $p$ safe
      // and create its transition rules.
    $Q_{eq}$ = $\{p' \in Q_{safe} \mid p^{-1}D = p'^{-1}D$ and not $p \not\parallel_S p'\}$
    $(u, v) = p$
  in
    case $Q_{eq}$ // $Q_{eq}$ may contain at most 1 element
    of $\{p'\}$ then replace all occurrences of $p$ in $M$ by $p'$
                     $Q_{temp}$ := $Q_{temp} \backslash \{p\}$
    of $\emptyset$ then
      $Q_{safe}$ := $Q_{safe} \cup \{p\}$
      $Q_{temp}$ := $Q_{temp} \setminus \{p\}$
      for $f \in F$ where $uf^{-1}D \neq \emptyset$ do
        $V'$ = $\{v' \mid v'^{-1}out_{p^{-1}S}(f) = \bot\}$
        fun $i(v')$ // where $v' \in V'$
          unique index $i$ s.t. $(ufi, vv')^{-1}S$ is functional
              // exists by Corollary 81 since
              // $D$ is defined by a DTTA.
        end
        fun $state(v')$ // where $v' \in V'$
            $(ufi(v'), vv')$
        end
        fun $call(v')$ // where $v' \in V'$
            $state(v')\langle x_{i(v')}\rangle$
        end
      in
        $Q_{temp}$ := $Q_{temp} \cup \{state(v') \mid v' \in V'\}$
        $rhs(p, f)$ := $out_{p^{-1}S}(f)$ $[v' \leftarrow call(v') \mid v' \in V']$
      end
    end
  end
in
    while $Q_{temp} \neq \emptyset$ do
      $p$ = $min(Q_{temp})$
    in
      $integrate\text{-}state(p)$
    end
    return $(M, D)$
end
```

Figure 4.5: Learning algorithm of DTops with top-down inspection.

Action	Q_{safe}	rhs	Q_{temp}
Initialization	$\emptyset$	$\emptyset$	$\boldsymbol{(\varepsilon, \mathsf{P1}); (\varepsilon, \mathsf{P2})}$
integrate-state $(\varepsilon, \mathsf{P1})$	$\boldsymbol{(\varepsilon, \mathsf{P1})}$	$\boldsymbol{(\varepsilon, \mathsf{P1})(\mathsf{P}(x_1, x_2)) \rightarrow (\mathsf{P2}, \mathsf{P1})\langle x_2 \rangle}$	$(\varepsilon, \mathsf{P2}); \boldsymbol{(\mathsf{P2}, \mathsf{P1})}$
integrate-state $(\varepsilon, \mathsf{P2})$	$(\varepsilon, \mathsf{P1})$ $\boldsymbol{(\varepsilon, \mathsf{P2})}$	$(\varepsilon, \mathsf{P1})(\mathsf{P}(x_1, x_2)) \rightarrow (\mathsf{P2}, \mathsf{P1})\langle x_2 \rangle$ $\boldsymbol{(\varepsilon, \mathsf{P2})(\mathsf{P}(x_1, x_2)) \rightarrow (\mathsf{P1}, \mathsf{P2})\langle x_1 \rangle}$	$\boldsymbol{(\mathsf{P1}, \mathsf{P2})}; (\mathsf{P2}, \mathsf{P1})$
integrate-state $(\mathsf{P1}, \mathsf{P2})$	$(\varepsilon, \mathsf{P1})$ $(\varepsilon, \mathsf{P2})$ $\boldsymbol{(\mathsf{P1}, \mathsf{P2})}$	$(\varepsilon, \mathsf{P1})(\mathsf{P}(x_1, x_2)) \rightarrow (\mathsf{P2}, \mathsf{P1})\langle x_2 \rangle$ $(\varepsilon, \mathsf{P2})(\mathsf{P}(x_1, x_2)) \rightarrow (\mathsf{P1}, \mathsf{P2})\langle x_1 \rangle$ $\boldsymbol{(\mathsf{P1}, \mathsf{P2})(\mathsf{A}(x_1)) \rightarrow \mathsf{A}((\mathsf{P1A1}, \mathsf{P2A1})\langle x_1 \rangle)}$ $\boldsymbol{(\mathsf{P1}, \mathsf{P2})(\#) \rightarrow \#}$	$(\mathsf{P2}, \mathsf{P1}); \boldsymbol{(\mathsf{P1A1}, \mathsf{P2A1})}$
integrate-state $(\mathsf{P2}, \mathsf{P1})$	$(\varepsilon, \mathsf{P1})$ $(\varepsilon, \mathsf{P2})$ $(\mathsf{P1}, \mathsf{P2})$ $\boldsymbol{(\mathsf{P2}, \mathsf{P1})}$	$(\varepsilon, \mathsf{P1})(\mathsf{P}(x_1, x_2)) \rightarrow (\mathsf{P2}, \mathsf{P1})\langle x_2 \rangle$ $(\varepsilon, \mathsf{P2})(\mathsf{P}(x_1, x_2)) \rightarrow (\mathsf{P1}, \mathsf{P2})\langle x_1 \rangle$ $(\mathsf{P1}, \mathsf{P2})(\mathsf{A}(x_1)) \rightarrow \mathsf{A}((\mathsf{P1A1}, \mathsf{P2A1})\langle x_1 \rangle)$ $(\mathsf{P1}, \mathsf{P2})(\#) \rightarrow \#$ $\boldsymbol{(\mathsf{P2}, \mathsf{P1})(\mathsf{B}(x_1)) \rightarrow \mathsf{B}((\mathsf{P2B1}, \mathsf{P1B1})\langle x_1 \rangle)}$ $\boldsymbol{(\mathsf{P2}, \mathsf{P1})(\#) \rightarrow \#}$	$(\mathsf{P1A1}, \mathsf{P2A1}); \boldsymbol{(\mathsf{P2B1}, \mathsf{P1B1})}$
integrate-state $(\mathsf{P1A1}, \mathsf{P2A1})$	$(\varepsilon, \mathsf{P1})$ $(\varepsilon, \mathsf{P2})$ $(\mathsf{P1}, \mathsf{P2})$ $(\mathsf{P2}, \mathsf{P1})$	$(\varepsilon, \mathsf{P1})(\mathsf{P}(x_1, x_2)) \rightarrow (\mathsf{P2}, \mathsf{P1})\langle x_2 \rangle$ $(\varepsilon, \mathsf{P2})(\mathsf{P}(x_1, x_2)) \rightarrow (\mathsf{P1}, \mathsf{P2})\langle x_1 \rangle$ $(\mathsf{P1}, \mathsf{P2})(\mathsf{A}(x_1)) \rightarrow \mathsf{A}(\boldsymbol{(\mathsf{P1}, \mathsf{P2})}\langle x_1 \rangle)$ $(\mathsf{P1}, \mathsf{P2})(\#) \rightarrow \#$ $(\mathsf{P2}, \mathsf{P1})(\mathsf{B}(x_1)) \rightarrow \mathsf{B}((\mathsf{P2B1}, \mathsf{P1B1})\langle x_1 \rangle)$ $(\mathsf{P2}, \mathsf{P1})(\#) \rightarrow \#$	$(\mathsf{P2B1}, \mathsf{P1B1})$
integrate-state $(\mathsf{P2B1}, \mathsf{P1B1})$	$(\varepsilon, \mathsf{P1})$ $(\varepsilon, \mathsf{P2})$ $(\mathsf{P1}, \mathsf{P2})$ $(\mathsf{P2}, \mathsf{P1})$	$(\varepsilon, \mathsf{P1})(\mathsf{P}(x_1, x_2)) \rightarrow (\mathsf{P2}, \mathsf{P1})\langle x_2 \rangle$ $(\varepsilon, \mathsf{P2})(\mathsf{P}(x_1, x_2)) \rightarrow (\mathsf{P1}, \mathsf{P2})\langle x_1 \rangle$ $(\mathsf{P1}, \mathsf{P2})(\mathsf{A}(x_1)) \rightarrow \mathsf{A}((\mathsf{P1}, \mathsf{P2})\langle x_1 \rangle)$ $(\mathsf{P1}, \mathsf{P2})(\#) \rightarrow \#$ $(\mathsf{P2}, \mathsf{P1})(\mathsf{B}(x_1)) \rightarrow \mathsf{B}(\boldsymbol{(\mathsf{P2}, \mathsf{P1})}\langle x_1 \rangle)$ $(\mathsf{P2}, \mathsf{P1})(\#) \rightarrow \#$	$\emptyset$

that if N is a $\text{DTopI}_{\text{td}}(D)$, $\tau = [\![N]\!]$, S is a characteristic sample for τ and $min(\tau) = (M_\tau, D)$, then $learn_D(S)$ is equal to $repcan(\tau)$.

To this end, we will show that at each intermediary step of the algorithm, we will learn a "partially unfolded" version of $repcan(\tau)$: before calling *integrate-state* on a pair p, all states $p' < p$ should be in Q_{safe}, all pairs $(\varepsilon, v) \in minp^+$ or $Succ_\tau(Q_{safe})$ should have appeared in Q_{temp}, but all such pair smaller than p should already be integrated. This leads to a transducer that has some definitive states in Q_{safe}, some unexplored temporary states in Q_{temp}, and some calls in ax and rhs possibly pointing to a safe state or a temporary state, depending on their lexical order relative to p.

To formalize this notion of partially unfolded rules, we first define what pairs should be replaced, what pairs should still be unexplored, and we define the *p-truncated* version of $repcan(\tau)$.

Definition 106. *Let N be a $\text{DTopI}_{td}(D)$, $\tau = [\![N]\!]$, and p an aligned pair of τ. For $p' \in minp^+(\tau)$, we call $minrep^p_\tau(p')$ the p-truncated representative of p':*

- *$minrep^p_\tau(p') = p''$ the unique element of $minp(\tau)$ such that $p'' \equiv_\tau p'$ if $p' < p$,*
- *$minrep^p_\tau(p') = p'$ itself if $p' \geqslant p$*

Definition 107. *Let N be a $\text{DTopI}_{td}(D)$, $\tau = [\![N]\!]$, $repcan(\tau) = (Q, F, G, ax, rhs)$, and p an aligned pair of τ. We define the p-truncated form of $repcan(\tau)$, $repcan_p(\tau) = (Q_{safe(p)} \cup Q_{temp(p)}, F, G, ax_p, rhs_p)$, where:*

- *The p-truncated safe states $Q_{safe(p)} = \{p' \in Q \mid p' < p\}$*
- *The p-truncated temporary states*
  $Q_{temp(p)} = \big(\{(\varepsilon, v) \mid v^{-1}ax = q\langle x_0\rangle\} \cup Succ_\tau(Q_{safe(p)})\big) \cap \{p' \mid p' \geqslant p\}$
- *The p-truncated axiom*
  $ax_p = ax\ [v \leftarrow minrep^p_\tau(\varepsilon, v)\langle x_0\rangle \mid v^{-1}ax = q\langle x_0\rangle]$
- *The p-truncated rules rhs_p are defined so that if $p' = (u', v') \in Q_{safe(p)}$, and f a letter such that $rhs(p', f)$ is defined, then*

$$\begin{aligned} rhs_p(p', f) = rhs(p', f)\ [v'' \leftarrow & minrep^p_\tau(u'fi, v'v'')\langle x_i\rangle \\ & \mid v''^{-1}rhs(p', f) = q\langle x_i\rangle] \end{aligned}$$

Note that the p-truncated form "develops" as p grows, to finally become $repcan(\tau)$ itself if $p > max(minp^+(\tau))$.

Corollary 108. *Let N be a $\text{DTopI}_{td}(D)$, $\tau = [\![N]\!]$, $N_\tau = (M_\tau, D)$ the canonical form of N, p semantically aligned for τ such that $p > max(minp^+(\tau))$. Then $repcan_p(\tau) = repcan(\tau)$.*

Proof. By construction, if $p > max(minp^+(\tau))$, for all pairs $p' \in minp^+(\tau)$, $minrep^p_\tau(p') = minp_\tau(p')$. Furthermore, $Q_{safe(p)} = minp(\tau)$, $Q_{temp(p)} = \emptyset$ hence $repcan_p(\tau)$ and $repcan(\tau)$ have same states. Finally, the only difference between the definition of the axiom and rules of $repcan_p(\tau)$ and $repcan(\tau)$ is that the former uses $minrep^p_\tau$ and the latter uses $minp_\tau$. Since they are both identical for all pairs of $minp^+(\tau)$, we have that $repcan_p(\tau)$ and $repcan(\tau)$ have same axioms and rules. □

We prove the correctness of $learn_D$ if the sample S is characteristic. To this end, we show that right before each call to *integrate-state*(p), the transducer M created by $learn_D$ is exactly M_p.

Proposition 109. *Let N be a* $\text{DTopI}_{td}(D)$*, $\tau = [\![N]\!]$. Then $learn_D(S) = repcan(\tau)$.*

Proof. We prove the following invariant: if $Q_{temp} \neq \emptyset$ and $p = min(Q_{temp})$, then $Q_{safe} = Q_{safe(p)}$, $Q_{temp} = Q_{temp(p)}$, $ax = ax_p$ and $rhs = rhs_p$.

After the initialization, $Q_{safe} = \emptyset$ and $Q_{temp} = \{(\varepsilon, v) \mid v^{-1}out_S(\varepsilon) = \bot\}$. Since S is characteristic, this means $Q_{temp} = \{(\varepsilon, v) \mid v^{-1}out_\tau(\varepsilon) = \bot\}$. If $Q_{temp} \neq \varepsilon$, we call $p = min(Q_{temp})$. p is the smallest aligned pair of τ. This means that $Q_{safe(p)} = minp \cap \{p' \mid p' < p\}$ is empty, and therefore $Q_{safe(p)} = Q_{safe}$. This gives us that $Q_{temp(p)} = \{(\varepsilon, v) \mid v^{-1}out_\tau(\varepsilon) = \bot\} \cap \{p' \mid p' \geqslant p\}$. Since all aligned pairs are bigger than p, we have $Q_{temp(p)} = Q_{temp}$.
Since p is the smallest aligned pair, $minrep^p_\tau$ is the identity function. This means that $ax_p = out_\tau(\varepsilon)\ [v \leftarrow (\varepsilon, v)\langle x_0\rangle \mid v^{-1}out_\tau(\varepsilon) = \bot]$. The current axiom in learn is $ax = out_S(\varepsilon)\ [v \leftarrow (\varepsilon, v)\langle x_0\rangle \mid v^{-1}out_S(\varepsilon) = \bot]$. Since S is characteristic, $ax_p = ax$. As for rules, none have been created yet, which means $rhs = rhs_p = \emptyset$.

For the inductive case, we consider $p = (u, v)$ the minimal element of Q_{temp}. We have $Q_{safe} = Q_{safe(p)}$, $Q_{temp} = Q_{temp(p)}$, $ax = ax_p$ and $rhs = rhs_p$. We call the new values after *integrate-state*(p) Q'_{safe}, Q'_{temp}, ax' and rhs'. If $Q'_{temp} \neq \emptyset$, we call p' its minimum. p is added to Q_{safe} if and only if for all $p'' \in Q_{safe}$, $p \not\Vdash_S p''$. Since S is characteristic, $Q_{safe} \subseteq minp(\tau)$ and $p \in minp^+(\tau)$, this means that for all $p'' \in Q_{safe(p)}$, $p \not\equiv_\tau p''$. Hence, p is added to Q_{safe} if and only if $p \in minp(\tau)$. For the same reason, states are added to Q_{temp} if and only if $p \in minp(\tau)$. If S is characteristic, then $V' = \{v' \mid v'^{-1}out_\tau(f) = \bot\}$, and for each $v' \in V'$, then $i(v')$ is the unique i such that $(ufi, vv')^{-1}\tau$ is functional, i.e. $i(v') = ind_\tau(p, f, v')$. This means that the new temporary states of Q'_{temp} are $\{state(v') \mid v' \in V'\} = Succ_\tau(p)$. In all cases p is removed from Q_{temp}. This means that regardless of weather p was added or not, $Q'_{safe} = minp(\tau) \cap \{p'' \mid p'' \leqslant p\}$ and $Q'_{temp} = \{(\varepsilon, v) \mid v^{-1}out_\tau(\varepsilon) = \bot\} \cap \{p'' \mid p'' > p\}$. This means that Q'_{safe} and Q'_{temp} are $Q_{safe(p'')}$ and $Q_{temp(p'')}$ for some pair p'' right after p. Since $p' = min(Q_{temp})$, there is no element of $minp^+(\tau)$ between p and p'. Thus, $Q'_{safe} = Q_{safe(p')}$, and $Q'_{temp} = Q_{temp(p')}$.

If p is not a new state, then $minrep^p_\tau$ is different from $minrep^{p'}_\tau$ only for p, which has to be replaced by the only $p'' \in Q_{safe}$ such that $p'' \equiv_\tau p$. As *integrate-state*(p) replaces every state call $p\langle x_i\rangle$ by $p''\langle x_i\rangle$, we have $ax' = ax_{p'}$ and $rhs' = rhs_{p'}$. However, if p is a new state, then $minrep^{p'}_\tau = minrep^p_\tau$, and thus $ax' = ax_{p'}$, but new rules have to be added. *integrate-state*(p) adds new rules. For $p = (u, v)$, we have that for each f such that $uf^{-1}D \neq \emptyset$, we create the rule $rhs'(p, f) = out_{p^{-1}S}(f)\ [v' \leftarrow call(v') \mid v' \in V']$. As previously mentioned, $V' = \{v' \mid v'^{-1}out_\tau(f) = \bot\}$, and $i(v') = ind_\tau(p, f, v')$. This means that $rhs'(p, f) = out_{p^{-1}\tau}(f)\ [v' \leftarrow (ufi, vv')\langle x_i\rangle \mid (ufi, vv') = succ_\tau(p, f, v'), v' \in V']$. For all (ufi, vv') in $Succ_\tau(p)$, we know that $(ufi, vv') > p$, and thus $p'' = minrep^{p'}_\tau(p'')$. Hence, $rhs' = rhs_{p'}$.

It remains to show that the last step that eventually empties Q_{temp} leads to $repcan(\tau)$. If Q_{temp} starts as $\emptyset$, this means $out_S(\varepsilon)$ has no $\bot$-leaf. Since S is characteristic, this means $out_\tau(\varepsilon)$ has no $\bot$-leaf. This is only possible if τ is a constant transduction that sends all trees of D to the same image t. In this case, $learn_D(S)$ produces a transducer with no states and no rules, and an axiom $ax = t$, which is indeed the canonical form of τ. In all other cases, we consider p the last pair to be integrated by *integrate-state*(p). Since it is the last considered pair, we have that $p = max(minp^+(\tau))$. As seen in this proof, after this last *integrate-state*(p), we have $learn_D(S) = repcan_{p'}(\tau)$, where p' is the pair right after p in lexical order. Thus, Corollary 108 gives us that $learn_D(S) = repcan(\tau)$. □

4.10 Remaining Proofs

4.10.1 Trimming a DTopI

We describe the construction of a trimmed compatible DTopI from a DTopI. The existence of such a DTopI is argued in Proposition 51.

Proposition 51. *Any* DTopI$_{reg}$ *is equivalent to some trimmed* DTopI$_{reg}$ *and any* DTopI$_{td}$ *is equivalent to some trimmed* DTopI$_{td}$.

Proof. Let $N = (M, D)$ be a DTopI$_{\mathrm{reg}}$, $M = (Q, F, G, ax, rhs)$. We call $D' = dom([\![N]\!])$, and consider a trimmed nondeterministic tree automaton $A = (F, P, P_I, \Delta)$ such that $[\![A]\!] = D'$.

We want to find for each state $q \in Q$ the set P_q of states of P reached by q:

$$P_q = \{p \in P \mid \exists(u, v) \mid u \sim_q v \text{ and } u \text{ reaches } p\}$$

.

We can compute P_q recursively, as shown by this equivalent mutually recursive definition. The sets P_q are the smallest subsets of P such that:

- if $q\langle x_0\rangle$ occurs in ax, then $P_I \subseteq P_q$.

- if $p \in P_q$, $p \xrightarrow{f} (p_1, ..., p_i, ..., p_n)$ a rule of A, and $q'\langle x_i \rangle$ occurs in $rhs(q, f)$, then $p_i \in P_{q'}$.

The equivalence between those two definitions can be proven by recursion on the length of the input path u required to access a state $q \in Q$ at the same time as a state $p \in P$. For $u = \varepsilon$, a pair (u, v) is aligned in q if and only if $v^{-1}ax = q\langle x_0 \rangle$. Conversely, ε reaches exactly P_I the set of initial states of A. For the recursion, $p \in P_q$ means that there exists a pair (u, v) such that $u \sim_q v$ and u reaches p. If $q'\langle x_i \rangle$ occurs in $rhs(q, f)$, there exists v' such that $v'^{-1}rhs(q, f) = q'\langle x_i \rangle$. This means $ufi \sim_{q'} vv'$. Meanwhile in A, if there is a rule $p \xrightarrow{f} (p_1, ..., p_i, ..., p_n)$, and u reaches p, then ufi reaches p_i, which means $p_i \in P_{q'}$.

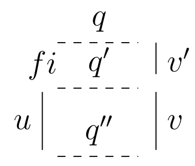
From there, selecting useful states and rules is immediate. A state q is useful if and only if $P_q \neq \emptyset$. A rule $rhs(q, f)$ is useful if and only if there exists a state $p \in P_q$ with a rule $p \xrightarrow{f} (p_1, ..., p_i, ..., p_n)$. □

4.10.2 Origins of Output Constructors

We want to prove Proposition 52, that states that when computing an image of a DTop, each constructor of the output tree is produced either by the axiom or when rewriting the subtree at a unique input path.

In order to do this, we will need an alternative characterization of syntactic alignments. It deals with the fact that the definition of judgements $u \sim_{q'}^{q} v$ extends paths u to the right by repeated concatenation. The alternative characterization is given by the following equivalent judgements $u \dot{\sim}_{q'}^{q} v$ that repeatedly concatenates to the left instead:

$$\frac{true}{\varepsilon \dot{\sim}_{q}^{q} \varepsilon} \qquad \frac{rhs(q, f) \models v'q'\langle x_i \rangle \quad u \dot{\sim}_{q''}^{q'} v}{fiu \dot{\sim}_{q''}^{q} v'v}$$



Lemma 110. *$u \sim_{q'}^{q} v$ if and only if $u \dot{\sim}_{q'}^{q} v$.*

Proof. We can show that $u \sim_{q'}^{q} v$ if and only if there exists a sequence $q_0, ..., q_n$ such that $q_0 = q$, $q_n = q'$, $u = f_0 i_0 ... f_{n-1} i_{n-1}$, $v = v_0 ... v_{n-1}$, and for all j from 0 to $n - 1$, $rhs(q_j, f_j) \models v_j q_{j+1} \langle x_{i_j} \rangle$. This comes from a simple proof by induction. The first definition adds an additional state after q_n, an input letter after $f_{n-1} i_{n-1}$, and output path after v_{n-1}. The second definition adds an additional state before q_0, an input letter before $f_0 i_0$, and output path after v_0. □

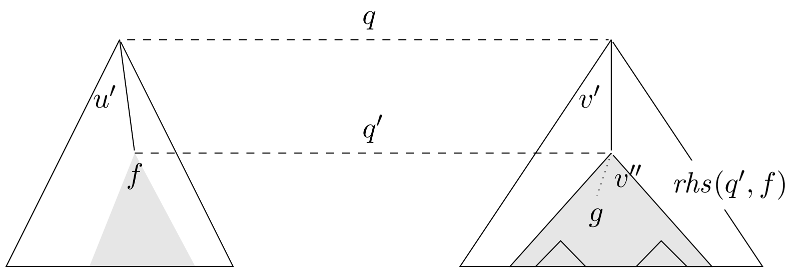


Figure 4.6: Constructor g at the output node $v = v'v''$ is produced by $[\![M]\!]_q$ from the f-rooted subtree located at node u' of the input tree. This is justified by the syntactic alignment $u' \sim^q_{q'} v'$ and $rhs(q', f) \models v''g$.

We next show that any constructor of an output tree produced by $[\![M]\!]_q$ comes from a syntactic alignment starting in state q. This is illustrated in Fig. 4.6 and formalized in Lemma 111.

Lemma 111. *Let M be a* DTOP *and $s \in dom([\![M]\!]_q)$ an input tree for some state q of M. Then for any output path v and letter $g \in G$ such that $[\![M]\!]_q(s) \models vg$ there exist a decomposition $v = v'v''$, an input path $u'f$, and a state q' such that:*

- *$s \models u'f$,*
- *$u' \sim^q_{q'} v'$, and*
- *$rhs(q', f) \models v''g$.*

Proof. The proof is by induction on the structure of s. Let $s = f'(s_1, \dots, s_k)$ for some $f'^{(k)} \in F$ and trees $s_1, \dots, s_k$. The definition of $[\![M]\!]_q$ yields:

$$[\![M]\!]_q(s) = rhs(q, f')[\tilde{q}\langle x_j\rangle \leftarrow [\![M]\!]_{\tilde{q}}(s_i) \mid \tilde{q} \in Q, 1 \le i \le k].$$

- Case $rhs(q, f') \models vg$. In this case, we choose $u' = v' = \varepsilon$, $f = f'$, and $q' = q$. Clearly, $s \models u'f$, $u' \sim^q_q v'$, and $rhs(q', f) \models v''g$.
- Case $rhs(q, f') \models v'_0\tilde{q}\langle x_i\rangle$ for some decomposition $v = v'_0v''_0$, $1 \le i \le k$, and state $\tilde{q}'$. Hence, ${v'_0}^{-1}[\![M]\!]_q(s) = [\![M]\!]_{\tilde{q}}(s_i)$, so that $s_i \in dom([\![M]\!]_{\tilde{q}})$ and $[\![M]\!]_{\tilde{q}}(s_i) \models v''_0g$. By induction hypothesis, there exist a decomposition $v'' = v'_1v''$, an input path u'_1f, and a state q' such that $s_i \models u'_1f$, $u'_1 \sim^{\tilde{q}}_{q'} v'_1$, and $rhs(q', f) \models v''g$. Hence, $s \models f'iu'_1f$ so we can choose $u' = f'iu'_1$ in order to satisfy the first condition $s \models u'f$. Lemma 110 yields $u'_1 \dot{\sim}^{\tilde{q}}_{q'} v'_1$, so that we can apply the inference rule:

$$\frac{rhs(q, f') \models v'_0\tilde{q}\langle x_i\rangle \qquad u'_1 \dot{\sim}^{\tilde{q}}_{q'} v'_1}{f'iu'_1 \dot{\sim}^{q}_{q'} v'_0v'_1}$$

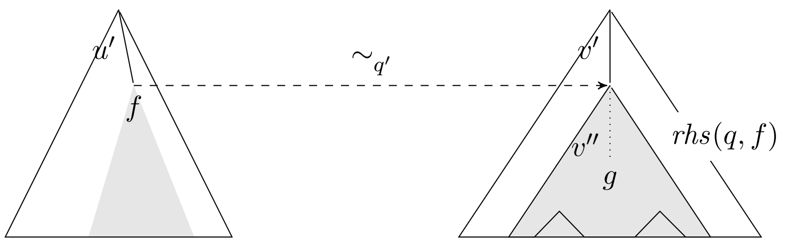


Figure 4.7: The f-subtree at input path u' produces the g-constructor at output path $v'v''$ by M. This is justified by the syntactic alignment starting with the axiom $u' \sim_{q'} v'$ and $rhs(q',f) \models v''g$.

Lemma 110 the other way around yields $u' \sim^{q}_{q'} v'_0 v'_1$, so we can choose $v' = v'_0 v'_1$ to show the second condition. The third condition $rhs(q',f) \models v''g$ is also satisfied as shown above.

□

We next lift the previous Lemma to how a constructor of the output tree may be produced by $[\![M]\!]$ instead of $[\![M]\!]_q$. There are two cases. Either the output constructor is produced by the axiom, or else it has its origin at some node of the input tree, as illustrated in Fig. 4.7 and formalized in the next proposition.

Proposition 52. *Let M be a* DTop *and $s \in dom(M)$ an input tree. Then for any output path vg such that $[\![M]\!](s) \models vg$, either $ax \models vg$ or there exist a unique decomposition $v = v'v''$, an input path $u'f$, and a state q' such that:*

- *$s \models u'f$,*
- *$u' \sim_{q'} v'$, and*
- *$rhs(q',f) \models v''g$.*

Proof. If $ax \models vg$ then the first case is satisfied. Otherwise, by Definition of $[\![M]\!](s)$, the axiom produces some call $q\langle x_0\rangle$ at a prefix of v. This means that there is a decomposition $v = v'_0 v''_0$ such that $ax \models v'_0 q\langle x_0\rangle$, ${v'_0}^{-1}[\![M]\!](s) = [\![M]\!]_q(s)$, and that $[\![M]\!]_q(s) \models v''_0 g$. We can now apply Lemma 111 to $[\![M]\!]_q(s) \models v''_0 g$, which shows that there exist a decomposition $v''_0 = v'_1 v''$, an input path $u'f$, and a state q' such that $s \models u'$, $u' \sim^{q}_{q'} v'_1$ and $rhs(q',f) \models v''g$. In this case, we can apply the following inference rule:

$$\frac{ax \models v'_0 q\langle x_0\rangle \qquad u' \sim^{q}_{q'} v'_1}{u' \sim_{q'} v'_0 v'_1}$$

With $v' = v'_0 v'_1$ we have the second condition $u' \sim_{q'} v'$, while the first condition $s \models u'$ and the third condition $rhs(q',f) \models v''g$ were show already above. □

4.11 Conclusions

Deterministic top-down tree transformations can be semantically characterized by studying semantic alignment, i.e. the dependencies between input and output paths when producing the output as early as possible. This notion of semantically aligned pairs allows for a definition of residual transformations. From there, there exists a Myhill-Nerode theorem on DTops. The normal form of [Engelfriet et al., 2009] coincides with the minimal normal form that comes from our Myhill-Nerode theorem. This normal form can be inferred by a Gold-style learning algorithm, polynomial in data and time.

For an extension of this result to wider classes, a first step would be to allow for DTops with regular domains: if semantic alignments and the notion of compatible and earliest transducers extend to regular inspections, the Myhill-Nerode theorem presented here relies on properties that would not extend to top-down tree transformations with regular domains. Another wider still interesting and robust class are top-down tree transducers with regular look-ahead [Engelfriet, 1977]. Such a transducer is allowed to first execute a bottom-up finite-state relabeling over the input tree, and then run the top-down translation on the relabeled tree. This extension, however, is more ambitious, for two main reasons. First, no normal form is known on this class. For string transducers with look-ahead [Elgot and Mezei, 1965], this normal form is composed of a look-ahead as coarse as possible, paired with the minimal transducer on the relabelling. We know this method cannot extend to tree transducers as is, since there is not a unique look-ahead as coarse as possible in the general case. Furthermore, even if a normal form is found, Gold-style learning algorithms encounter particular hurdles when dealing with look-ahead. For the learning problem on string transducers with look-ahead presented in Chapter 6, the normal form provided by the Myhill-Nerode algorithm is difficult to infer in polynomial time: polynomial inference is possible for another, different, more ad-hoc normal form.

As another possible avenue of extension, most Gold-style learning algorithms can be used as core in an interactive learner in Angluin-style [Angluin, 1987], similar to [Carme et al., 2007]. One might wonder if the algorithm presented in this chapter can be changed into an Angluin-style learning algorithm, with a polynomial number of learner-teacher interactions.

Chapter 5

Learning Top-Down Tree Transducers with Regular Inspection

Abstract. We study the problem of how to learn top-down tree transducers with regular domain inspection (DTopI$_{\text{reg}}$) from a finite sample of input-output examples. An Rpni style learning algorithm that solves this problem in Gold's model with polynomial resources was given in the previous chapter, but restricted to the case of path-closed regular domains. In this chapter, we show that this restriction can be removed. For this, we present a new normal form for DTopI$_{\text{reg}}$ by extending the Myhill-Nerode theorem for DTop to regular domain inspections in a nontrivial manner. The Rpni style learning algorithm can also be lifted but becomes more involved in the process. [a]

[a] This chapter extends on a paper presented at ICGI 2016.

Contents

5.1 Introduction **102**
5.2 Running Example **102**
5.3 Semantic Equivalence **104**
5.4 Unique Normal Forms **107**
5.5 Learning from Examples **113**
5.5.1 Characteristic Samples 113
5.5.2 Learning Algorithm 118

5.1 Introduction

We have seen in the previous chapter that DTops with regular inspection can be made compatible and earliest. For DTops with top-down inspection, this lead to a learning algorithms in Gold's model with polynomial resources, similary as for subsequential transducers. We recall that the domain inspection of the target transformation is given as a parameter. The main limitation of the previous approach is that the domain must be a path-closed regular tree language, i.e. it must be definable by a top-down deterministic tree automaton. This limitation is fine for Dtds or Xml Schema(s), but inhibiting for many Xml transformations, whose domain is given by a RelaxNG schema, since these are made to support general regular tree languages. Therefore, we study the question whether this restriction can be removed.

In this chapter, we present a learning algorithm for DTops with regular domain inspection ($\text{DTopI}_{\text{reg}}$), which generalizes on the learning algorithm for DTops from the previous chapter. In a first step, we show for any $\text{DTopI}_{\text{reg}}$ that its semantic equivalence has finite index. This requires a novel and nontrivial argument. Due to this result, we can obtain a new normal form for $\text{DTopI}_{\text{reg}}$ that we express with a new Myhill-Nerode theorem. We then lift the learning algorithm from [Lemay et al., 2010] so that it can account for regular domain inspection.

5.2 Running Example

Throughout this chapter, we consider an example for a $\text{DTopI}_{\text{reg}}$, which converts articles which may either be physical or digital. A digital article has a website, and an Url, while a physical article has a *collection*, and a *book* to which it belongs. Furthermore, each article has a full identifier, that contains its type (digital/physical), but also the full information of the Url or book.

In order to ensure a finite alphabet, we will represent websites and collections by letters $A \in \{a, a'\}$. The additional information for the Url or book will be represented by the letters $B \in \{b, b'\}$. A digital article then has the form:

$$\mathit{article}(\mathit{fullid}(\mathit{digital}, A, B), \mathit{website}(A), \mathit{url}(A, B))$$

and a physical article the form:

$$\mathit{article}(\mathit{fullid}(\mathit{physical}, A, B), \mathit{collection}(A), \mathit{book}(A, B)).$$

This representation of articles contains some redundancy, which typically happens when Xml documents are published by some database. In particular, the website can be infered from the Url, and the collection from the book.

The set of all such articles is finite and thus regular, but it is not path-closed. We consider the transformation τ_{ref} that will map each such articles to its type and website or collection:

$$\tau_{\text{ref}} = \left[\begin{array}{lcl} \mathit{article}(\mathit{fullid}(\mathit{digital}, A, B), \mathit{website}(A), \mathit{url}(A, B)) & / & \mathit{digital}(A), \\ \mathit{article}(\mathit{fullid}(\mathit{physical}, A, B), \mathit{collection}(A), \mathit{book}(A, B)) & / & \mathit{physical}(A) \\ \quad \mid \quad A \in \{a, a'\}, B \in \{b, b'\} & & \end{array} \right]$$

We now define a DTop M_{ref} such that $\tau_{\text{ref}} \subseteq [\![M_{\text{ref}}]\!]$. This transducer has three states q_0, q_1, q_2, the axiom $ax = q_0\langle x_0\rangle$, and the rules:

$$\begin{array}{llll} (1) & q_0(\mathit{article}(x_1, x_2, x_3)) \to q_1\langle x_2\rangle & (4) & q_2(a) \to a \\ (2) & q_1(\mathit{website}(x_1)) \to \mathit{digital}(q_2\langle x_1\rangle) & (5) & q_2(a') \to a' \\ (3) & q_1(\mathit{collection}(x_1)) \to \mathit{physical}(q_2\langle x_1\rangle) & & \end{array}$$

As an example, we show the step-by-step computing of the image of a tree $s = \mathit{article}(\mathit{fullid}(\mathit{physical}, a', b), \mathit{collection}(a'), \mathit{book}(a', b))$:

$$\begin{array}{rcl} [\![M_{\text{ref}}]\!](s) & = & [\![M_{\text{ref}}]\!]_{q_0}(s) = [\![M_{\text{ref}}]\!]_{q_1}(\mathit{collection}(a')) \\ & = & \mathit{physical}([\![M_{\text{ref}}]\!]_{q_2}(a')) = \mathit{physical}(a') \end{array}$$

A regular domain inspection to $D_{\text{ref}} = \mathit{dom}(\tau_{\text{ref}})$ is needed, in order to restrict $[\![M_{\text{ref}}]\!]$ so that it becomes equal to τ_{ref}. Therefore, we consider the DTopI$_{\text{reg}}$ $N_{\text{ref}} = (M_{\text{ref}}, D_{\text{ref}})$, which indeed is earliest and compatible.

The semantic alignments introduced in Section 4.7 deserves closer scrutiny in the case of DTopI$_{\text{reg}}$, as several properties essential for top-down inspection do fail with regular inspection. Notably, if semantically aligned pairs can be seen as candidates to be syntactically aligned pairs in earliest DTopI$_{\text{reg}}$, there exists some semantically aligned pairs that are not syntactically aligned for some earliest transducers. This can be observed at the above example;

The following pairs are both semantically aligned for τ_{ref} and syntactically aligned for M_{ref}:

$$(\varepsilon, \varepsilon), \ (\mathit{article}2, \varepsilon), \ (\mathit{article}2\mathit{website}1, \mathit{digital}1).$$

However, the following pair is semantically aligned for τ_{ref} but not syntactically aligned in N_{ref}:

$$(\mathit{article}3\mathit{url}1, \mathit{digital}1)$$

This is since N_{ref} choses $\mathit{article}2\mathit{website}1$ to produce as origin to produce the output at $\mathit{digital}1$ rather than $\mathit{article}3\mathit{url}1$. So there exists some DTopI$_{\text{reg}}$ defining τ_{ref} for which $(\mathit{article}3\mathit{url}1, \mathit{digital}1)$ is syntactically aligned.

Finally, the case of the pair $(\mathit{article}1, \varepsilon)$ is particularly remarquable. This pair is semantically aligned, but its residual cannot be computed by any DTopI, so it is neither syntacally aligned for N_{ref} nor for any other DTopI defining τ_{ref}. To see this, note that:

$$(\mathit{article}1, \varepsilon)^{-1}\tau_{\text{ref}} = \left[\begin{array}{c} \mathit{fullid}(\mathit{digital}, A, B)/\mathit{digital}(A), \\ \mathit{fullid}(\mathit{physical}, A, B)/\mathit{physical}(A) \\ \mid \quad A \in \{a, a'\}, B \in \{b, b'\} \end{array} \right]$$

The problem is A is a brother of *digital* and respectively *physical* and not a descendant, so after having output *digital* or *physical*, a top-down transducer cannot output the A.

Most results on normal forms for DTopIs with top-down inspection in the previous chapter are based on the existence and uniqueness of semantic successors (see Definition 80). For a DTopI$_{\text{reg}}$, however, these successors may *neither* be unique, nor even exist. For example, consider τ_{ref} and the semantically aligned pair $(\varepsilon, \varepsilon)$. Due to the redundancy, there are three semantic successors:

$$Succ_{\tau_{\text{ref}}}((\varepsilon, \varepsilon), article, \varepsilon) = \{(article1, \varepsilon), (article2, \varepsilon), (article3, \varepsilon)\}$$

Conversely, for the pair $(article1, \varepsilon)$ there is not semantic successor:

$$Succ_{\tau_{\text{ref}}}((article1, \varepsilon), fullid, \varepsilon) = \emptyset$$

The problem is that the residual cannot be defined by any DTopI, so neither $(article1fullid1, \varepsilon)$, $(article1fullid2, \varepsilon)$, nor $(article1fullid3, \varepsilon)$ have functional residuals.

So what we learn from this example is that the class DTopI$_{\text{reg}}$ permits to define transformations whose input trees contain redundancies. Therefore, some output nodes may be given different origins in the input tree. As a consequence, one may be free to chose between semantic alignments when defining a DTopI$_{\text{reg}}$ for a given transformation. Furthermore, there may be useless semantic alignments, that cannot be used by any DTopI, as long as the output path of such a pair can be given some other origin in the input tree.

5.3 Semantic Equivalence

In order to obtain a Myhill-Nerode theorem, we wish to prove that the semantic equivalence $\equiv_{[\![N]\!]}$ has finite index. We know from Corollary 57 that the syntactic equivalence $\equiv_N$ has finite index. However, as seen in Example 5.2, there are more semantic than syntactic aligned pairs. What we can show nevertheless is that the number of additional pairs is bounded.

Theorem 112. *For any* DTopI$_{reg}$ N *the equivalence relation* $\equiv_{[\![N]\!]}$ *has finite index.*

Proof. By Proposition 69, we can assume that N is earliest w.l.o.g. Therefore, $\equiv_N$ is included in $\equiv_{[\![N]\!]}$ as shown by Lemma 77, but not necessarily vice versa (see Example 5.2). The syntactic congruence $\equiv_N$ has finite index by Corollary 57, so we have to show that the semantic congruence $\equiv_{[\![N]\!]}$ can add at most finitely many equivalence classes to $\equiv_N$.

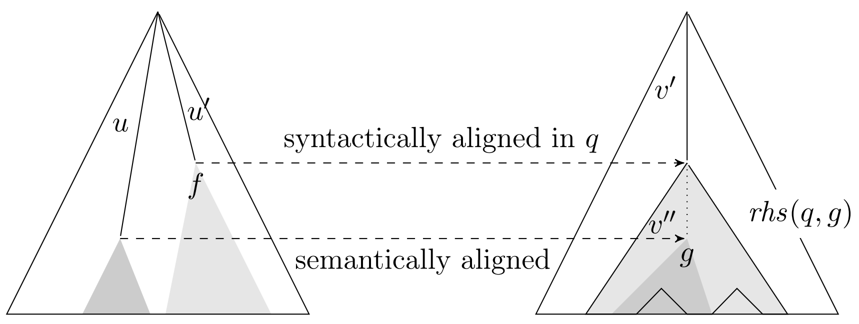


Figure 5.1: Input path u' that produces the node at output path $v = v'v''$.

Imagine a semantically aligned pair $p = (u, v)$ that is not syntactically aligned. We now consider a tree s such that $s \models u$. We consider the syntactic alignment that produce the node under v when the transducer N computes $[\![N]\!](s)$. This is the alignment $p' = (u', v')$ where the node under v is not produced yet, but will be after reading the node under u' in s. Formally, this means p' syntactically aligned in state q, such that $u'^{-1}s = f(t_1...t_n)$, $v'^{-1}[\![N]\!](s) = g(t'_1...t'_m)$, v' is a prefix of v such that $v = v'v''$, and $rhs(q, f) \models v''g$ (see Proposition 52).

We will start by asserting that in this scenario, u is not a prefix of u', nor the other way around: u and u' are disjoint. From there, we note that what is under v in the output depends functionally on what is below u and u' in the input. Since u and u' are two disjoint paths, this would be impossible in a path-closed domain. However, in a regular domain, these configurations are still possible. We will show that there is a finite amount of partial functions possible for $p^{-1}\tau$.

To prove that u and u' are disjoint, we start by proving that u' cannot be equal to u, then that neither can be a strict prefix of the other.

u equals u'. Assume that $u' = u$. We have $v^{-1}out_\tau(u) = \bot$, and $v'^{-1}out_\tau(u) = \bot$. Since v' is a prefix of v, this implies that $v' = v$. Hence, $p = p'$. This means that p is syntactically aligned, which is in contradiction with our assumption.

u is a prefix of u'. Suppose that u is a strict prefix of u'. Then by the recursive definition of syntactic alignments, there exists a syntactically aligned pair (u, v'') for some prefix v'' of v', and therefore, of v. Since we supposed N earliest, (u, v'') is semantically aligned, which means $v''^{-1}out_\tau(u) = \bot$. Since (u, v) is semantically aligned, we also have $v^{-1}out_\tau(u) = \bot$. This means that $v'' = v$, and thus that p is syntactically aligned, which is in contradiction with our assumption.

u' is a prefix of u. Suppose that u' is a strict prefix of u. Hence $u'f$ is a prefix of u too. From the assumption that $rhs(q, f) \models v''g$, we have that

$out_\tau(u'f) \models v'v''g$. Since $u'f$ is a prefix of u, we have $out_\tau(u) \models vg$. This means that $v^{-1}out_\tau(u) \neq \bot$, which is in contradiction with the assumption that p is semantically aligned.

From that we conclude that u and u' are disjoint.

We now study the residuals under p semantic alignment and p' syntactic alignment. By definition, $p^{-1}[\![N]\!]$ is functional. By Lemma 55, $p'^{-1}[\![N]\!]$ is functional. However, v' is a prefix of v. This means that if $[\![N]\!](s) \models v$, then $v^{-1}[\![N]\!](s)$ depends functionally on what is under u and u' (see Fig 5.1). This is not a contradiction in itself (as seen in Example 5.2), but it limits what the residual $p^{-1}[\![N]\!]$ can be. Indeed, if we replace the subtree under u in s with another one without changing the subtree under u', then $v^{-1}[\![N]\!](s)$ does not change. We then consider the tree that we can safely substitute under u in s with another one without changing the subtree under u'. The idea goes as follow: consider A the nondeterministic automaton of states in R that recognizes $dom([\![N]\!])$. If an accepting run of A on s reaches state r after reading u, then we could replace the subtree under u in s by any tree in L_r without changing the subtree under u'. This means that $p^{-1}[\![N]\!]$ is constant over L_r. By extending this reasoning to all s such that $s \models u$, we have that $p^{-1}[\![N]\!]$ has at most one value per L_r. Note that the argument works both ways, which means that $v''^{-1}p'^{-1}[\![N]\!]$ has at most one value per L_r.

We now show that these values are members of a finite set of possible values. We remember our previous assumption: p is semantically aligned, and p' is syntactically aligned in state q. We know that $p'^{-1}[\![N]\!] = [\![N]\!]_q$. For every tree s such that $s \models u$ and $s \models u'$, we know that $p^{-1}[\![N]\!](s) = v''^{-1}p'^{-1}[\![N]\!](s)$. As we have seen earlier in the proof, these functions have at most one value per L_r. We choose one tree $t_r \in L_r$ for each r state of A. For a tree s such that $s \models u$ and $s \models u'$, for p the state labelling $u^{-1}s$ in the accepting run of s in A, $p^{-1}[\![N]\!](s) = v''^{-1}[\![N]\!]_q(t_r)$. We generalize this reasoning to all semantically aligned pairs that are not syntactically aligned: all semantic alignments define functions with at most one value per L_r with r state of A, and each value is of form $v''^{-1}[\![N]\!]_q(t_r)$, for some t_p, for some q, for some v''. There is a finite amount of q and t_r. Since v'' is a path of $[\![N]\!]_q(t_r)$, there is also a finite amount of v''.

To summarize, if p is semantically aligned, but not syntactically aligned in any state q, then $p^{-1}[\![N]\!]$ has at most one value per L_r (finite number of states). Each of these values is of form $v''^{-1}[\![N]\!]_q(t_r)$ (finite number of q, t_r and v''). This means there is only a finite number of options available for functions $p^{-1}[\![N]\!]$. Hence there are finitely many different $p^{-1}[\![N]\!]$. $\equiv_{[\![N]\!]}$ is of finite index. $\square$

5.4 Unique Normal Forms

We will define a unique normal form for any DTopI$_{\text{reg}}$. These will be compatible and earliest, as in the case of top-down inspection. But in the case of regular inspection, these conditions are no more sufficient, given that a DTopI$_{\text{reg}}$ can choose between several semantically aligned pairs, as illustrated by Example 5.2. We will show that we can still obtain a unique normal form by forcing this choice to be leftmost.

Example 113. *As shown in Example 5.2, τ_{ref} has semantically aligned pairs that are not syntactically aligned in N_{ref}: $p_1 = (article1, \varepsilon)$ and $p_3 = (article3, \varepsilon)$.*

We present here a new transducer N'_{ref}, that defines τ_{ref} by visiting the third child of article instead of the second. The axiom is $ax = q'_0\langle x_0\rangle$, and the rules:

$$\begin{array}{llll}
(1) & q'_0(article(x_1,x_2,x_3)) \to q'_1\langle x_3\rangle & (4) & q'_2(a) \to a \\
(2) & q'_1(url(x_1,x_2)) \to digital(q'_2\langle x_1\rangle) & (5) & q'_2(a') \to a' \\
(3) & q'_1(book(x_1,x_2)) \to physical(q'_2\langle x_1\rangle) & &
\end{array}$$

Since articles contain redundant information, both this DTopI *and N_{ref} define exactly τ_{ref}. Note that there is no* DTopI *that defines τ_{ref} and where p_1 is syntactically aligned: the residual $(article1, \varepsilon)^{-1}$, while functional, cannot be computed in a* DTop *manner.*

To settle this problem, we decide to fix the choice of syntactically aligned pairs to always pick the leftmost choice whenever possible.

Definition 114. *An earliest compatible* DTopI *is* leftmost *if for every syntactically aligned pair $p = (ufi, v)$, there is no $j < i$ such that the residual $(ufj, v)^{-1}\tau$ is definable by some* DTopI.

This means that whenever an earliest transducer can choose visiting two different sons for a part of its output, it always chooses the leftmost one to still offer a residual definable by a DTopI production. In Example 113, the DTopI N'_{ref} is not leftmost, as $(article3, \varepsilon)$ is syntactically aligned, but $(article2, \varepsilon)$ is definable by a DTopI: it is in fact $[\![N_{\text{ref}}]\!]_{q_1}$. However, N_{ref} is leftmost: even if $(article2, \varepsilon)$ is syntactically aligned, and $(article1, \varepsilon)$ is semantically aligned, $(article1, \varepsilon)^{-1}\tau_{\text{ref}}$ cannot be defined by a DTopI, which is clear from the fact that $(article1, \varepsilon)$ has no semantic successor.

To properly study and build leftmost transducers, we will define a notion of *left indexes and successors* to characterise which indexes lead to successors whose residual is definable by some a DTopI:

Definition 115. *Let $N = (M, D)$ a* DTopI*, and $\tau = [\![N]\!]$. Let $p = (u, v)$ be a semantically aligned pair of τ, $f \in F$ and v' a path of G. We define $Ind_\tau^{\text{DTop}}(p, f, v') = \{i \mid (ufi, vv')^{-1}\tau$ is definable by some* DTopI$\}$.

For triplets p, f, v' such that $Ind_\tau^{\mathrm{DTop}}(p, f, v') \neq \emptyset$, we also define the leftmost index $ind_\tau^{left}(p, f, v') = min\left(Ind^{\mathrm{DTop}}(p, f, v')\right)$, and the leftmost successor $succ_\tau^{left}(p, f, v') = (uf ind_\tau^{left}(p, f, v'), vv')$.

Since any residual $(ufi, vv')^{-1}\tau$ are also functional, it follows that $Ind_\tau^{\mathrm{DTop}}(p, f, v')$ is always a subset of $Ind_\tau(p, f, v')$. This subset is potentially $Ind_\tau(p, f, v')$ itself, but can also be smaller, and even empty. The following Lemma shows the link between $p^{-1}\tau$ being definable by a DTopI, and the emptiness of $Ind_\tau^{\mathrm{DTop}}(p, f, v')$.

Lemma 116. *Let $N = (M, D)$ be a* DTopI*, and $\tau = [\![N]\!]$. Let $p = (u, v)$ be a semantically aligned pair of τ. We have that $p^{-1}\tau$ is definable by some* DTopI *if and only if for all f such that $uf^{-1}dom(\tau) \neq \emptyset$, and v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \perp$, $Ind_\tau^{\mathrm{DTop}}(p, f, v') \neq \emptyset$.*

Proof. We first prove that if $p^{-1}\tau$ is definable by some DTopI, then for all f, v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \perp$, $Ind_\tau^{\mathrm{DTop}}(p, f, v') \neq \emptyset$. We consider a DTopI $N' = (M', u^{-1}dom(\tau))$ such that $[\![N']\!] = p^{-1}\tau$. Proposition 69 ensures we can choose N' to be compatible and earliest. We note $M' = (Q, F, G, ax, rhs)$. Since p is semantically aligned, $out_{p^{-1}\tau}(\varepsilon) = \perp$. Hence, by Lemma 68, the axiom ax is a simple state call $q\langle x_0\rangle$ for some $q \in Q$. Then, by Lemma 68, we have that for all f such that $uf^{-1}dom(\tau) \neq \emptyset$, the rule $rhs(q, f)$ is of form $out_{p^{-1}\tau}(f)\Phi$, where Φ replaces the $\perp$-leaves under paths v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \perp$ into state calls $q'\langle x_i\rangle$. We note that if $v'^{-1}rhs(q, f) = q'\langle x_i\rangle$, then (fi, v') is syntactically aligned in state q'. This means that if we were to replace ax by $q'\langle x_0\rangle$, and the inspection by $ufi^{-1}dom(\tau)$, we obtain a DTopI that describes $(fi, v')^{-1}[\![N']\!]$, i.e. $(ufi, vv')^{-1}\tau$. Hence, i is an element of $Ind_\tau^{\mathrm{DTop}}(p, f, v')$.

For the proof in the other direction, we assume that for all f, v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \perp$, $Ind_\tau^{\mathrm{DTop}}(p, f, v') \neq \emptyset$. We pick for each of these f, v' a compatible earliest DTopI $N_{f,v'} = (M_{f,v'}, uf ind_\tau^{\mathrm{left}}(p, f, v')^{-1}dom(\tau))$, such that $[\![N_{f,v'}]\!] = succ_\tau^{\mathrm{left}}(p, f, v')^{-1}\tau$. We note $M_{f,v'} = (Q_{f,v'}, F, G, ax_{f,v'}, rhs_{f,v'})$. As above, the axiom ax is a simple state call $q\langle x_0\rangle$ for some $q \in Q_{f,v'}$. We will note this particular state $q_{f,v'}$. We can use those DTopI and merge them together to form a DTopI that describes $p^{-1}\tau$. We create $N' = (M, u^{-1}\tau)$, and $M = (Q, F, G, ax, rhs)$, where:

- $Q = \bigcup_{f,v'} Q_{f,v'} \cup \{q_0\}$
- $ax = q_0\langle x_0\rangle$
- $rhs = \bigcup_{f,v'} rhs_{f,v'} \cup rhs_{q_0}$ where rhs_{q_0} is the set of rules
  $rhs(q_0, f) = out_{p^{-1}\tau}(f)[v'/q_{f,v'}\langle x_{ind_\tau^{\mathrm{left}}(p,f,v')}\rangle]$

By definition of the semantics of a DTopI, we would have $[\![N']\!]_{q_0} = [\![N']\!] = p^{-1}\tau$. □

We use these definition to characterise a *pre-canonical* form $can'(\tau)$ for a transformation τ. Note that in the general case, $can'(\tau)$ will not be a transducer, as it has a potentially infinite number of states and rules. However, we will show that if τ is recognized by a DTopI$_{\text{reg}}$, then $can'(\tau)$ is a leftmost earliest compatible DTopI$_{\text{reg}}$ that defines τ.

Definition 117. *Let τ be a transformation from $\mathcal{T}_F$ to $\mathcal{T}_G$. We define its* pre-canonical form $can'(\tau)$ *as the pair $(M, dom(\tau))$, where $M = (Q, F, G, ax, rhs)$ such that:*

- $Q = \{[p]_\tau \mid p^{-1}\tau$ *is definable by some* DTopI$\}$
- $ax = out_\tau(\varepsilon)[v/[(\varepsilon, v)]_\tau\langle x_0\rangle \mid v^{-1}out_\tau(\varepsilon) = \bot]$
- *For its rules: for all $p = (u, v)$ such that $[p]_\tau \in Q$ and $f \in G$ such that $uf^{-1}dom(\tau) \neq \emptyset$,*

$$rhs([p]_\tau, f) = out_{p^{-1}\tau}(f)[v'/[(ufi, vv')]_\tau\langle x_i\rangle \mid v'^{-1}out_{p^{-1}\tau}(f) = \bot \text{ and } i = ind_\tau^{left}(p, f, v')]$$

Note that if τ is defined by some DTopI$_{\text{reg}}$, Theorem 112 ensures that $\{[p]_\tau \mid p^{-1}\tau$ is definable by some DTopI$\}$ is finite, and Lemma 116 ensures that $ind_\tau^{\text{left}}(p, f, v')$ is well-defined in this definition. This means that $can'(\tau)$ is, itself, a DTopI$_{\text{reg}}$. Furthermore, if $dom(\tau)$ is a top-down domain, then $can'(\tau)$ is trimmed and already coincides with $min(\tau)$ as defined for DTopI$_{\text{td}}$. In the more general case, where $dom(\tau)$ is regular but not path-closed, however, $can'(\tau)$ might not be trimmed. Therefore, we define canonical transducers as the trimmed versions of precanonical transducers.

Definition 118. *Le N a* DTopI$_{reg}$*, where $\tau = [\![N]\!]$ and $can'(\tau) = (M, D)$, $M = (Q, F, G, ax, rhs)$. We define its* canonical form $can(\tau)$ *as:*

- $Q' = \{q \in Q \mid \exists(u, v)$ *syntactically aligned in* $q\}$
- $ax' = ax$
- $rhs' = rhs_{|Q' \times F}$

Note that this normal form comes to extend the notion of Section 4.8, as both definition coincide on DTopI$_{\text{td}}$.

We next show that $can(\tau)$ can serve as normal-form via a Myhill-Nerode type theorem. To show this, we will first prove for transformation τ definable by some DTopI$_{\text{reg}}$ that $can'(\tau)$ is an equivalent leftmost DTopI, and that the unique minimal leftmost DTopI$_{\text{reg}}$ defining τ is $can(\tau)$.

Proposition 119. *For any transformation τ definable by some* DTopI$_{reg}$*, $can'(\tau)$ is a* DTopI$_{reg}$ *defining τ that is compatible, earliest, and leftmost.*

Proof. Suppose that $\tau = [\![N]\!]$ for some DTopI$_{\text{reg}}$ $N = (M, D)$. By Theorem 112, $can'(\tau)$ has a finite number of states, since $\equiv_\tau$ has a finite index. Thus, $can'(\tau)$ is in fact a DTopI$_{\text{reg}}$. Let $can'(\tau) = N' = (M', dom(\tau))$. We will first prove that N' defines τ. Then, we will show it is also compatible, earliest, and leftmost.

The first step is to show that $[\![N']\!]_{[p]_\tau} = p^{-1}\tau$. We prove for the input tree $s = f(s_1, ..., s_n)$ that:

$$[\![N']\!]_{[p]_\tau}(s) = out_{p^{-1}\tau}(f)[v'/[\![N']\!]_{[(ufi,vv')]_\tau}(s_i) \mid v'^{-1}out_{p^{-1}[\![N]\!]}(f) = \bot \\ \text{and } i = ind^{\text{left}}(p, f, v')]$$

This is done by induction on s.

By induction hypothesis, $[\![N']\!]_{[(ufi,vv')]_\tau}(s_i) = (ufi, vv')^{-1}\tau(s_i) = v'^{-1}p^{-1}\tau(s)$. We can then prove $[\![N']\!]_{[p]_\tau}(s) = p^{-1}\tau(s)$. All paths v'' such that $out_{p^{-1}\tau}(f) \models v''$ are both in $p^{-1}\tau(s)$ and in $[\![N']\!]_{[p]_\tau}(s)$. Plus, for v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \bot$, $v'^{-1}[\![N']\!]_{[p]_\tau}(s) = v'^{-1}p^{-1}\tau(s)$.

We add the axiom on top of these production to show that $\tau = [\![N']\!]$. For a tree $s \in dom(\tau)$ we have that $[\![N']\!](s) = out_\tau(\varepsilon)[v/[\![N']\!]_{[(\varepsilon,v)]_\tau}(s) \mid v^{-1}out_\tau(\varepsilon) = \bot]$. We can then prove $[\![N']\!](s) = \tau(s)$. All paths v' such that $out_\tau(\varepsilon) \models v'$ are both in $\tau(s)$ and in $[\![N']\!](s)$. Plus, for v such that $v^{-1}out_\tau(\varepsilon) = \bot$, $v^{-1}[\![N']\!](s) = v^{-1}\tau(s)$.

To show that $[\![N']\!]$ is compatible and earliest, we prove that if a pair $p = (u, v)$ is syntactically aligned, then $u \sim_{[p]_\tau} v$. This can be proven by induction on the length of u. If $u = \varepsilon$, then $\varepsilon \sim_{[p']_\tau} v$ if and only if $v^{-1}ax = [p']_\tau\langle x_0\rangle$. From Definition 117, this means $v^{-1}out_\tau(\varepsilon) = \bot$, and that $[(\varepsilon, v)]_\tau = [p']_\tau$. If $u = u_0fi$, and $u_0 \sim_{[p_0]_\tau} v_0$, and $u \sim_{[p']_\tau} v$, then there exists v_0, v_1 such that $v_1 rhs'([p_0]_\tau, f) = [p']_\tau\langle x_i\rangle$. By induction, we know that $[(u_0, v_0)]_\tau = [p_0]_\tau$. From Definition 117, $v_1^{-1}rhs'([p_0]_\tau, f) = [p']_\tau\langle x_i\rangle$ means that $v_1^{-1}out_{(u_0,v_0)^{-1}\tau}(\varepsilon) = \bot$, and that $[(u_0fi, v_0v_1)]_\tau = [p']_\tau$.

From there, it follows immediately that N' is both compatible and earliest. If p and p' are syntactically aligned in the same state $[p'']_\tau$, then $[p]_\tau = [p']_\tau = [p'']_\tau$. This means $p^{-1}dom(\tau) = p'^{-1}dom(\tau)$. Hence, N' is compatible. Furthermore, if $p = (u, v)$ is a pair syntactically aligned, then $u \sim_{[p]_\tau} v$. From Definition 117, this means that $[p]_\tau$ is a class of $\equiv_\tau$. Hence, $out_{p^{-1}\tau}(\varepsilon) = \bot$. This means N' is earliest.

We can finally show that N' is leftmost almost by construction. If $p = (u'fi, v)$ is a pair syntactically aligned, that means $u \sim_{[p]_\tau} v$. By induction on syntactically aligned pairs, there is v', v'' such that $v = v'v''$, $u' \sim_{[(u',v')]_\tau} v'$, and $v''^{-1}rhs([p']_\tau, f) = [p]_\tau\langle x_i\rangle$. From Definition 117, i is the leftmost index for the pair $(uf, v'v'')$. This means that there is no j such that $j < i$ and $(ufj, v'v'')^{-1}\tau$ is definable by a DTopI$_{\text{reg}}$. This means that N' is leftmost. □

Now that we have shown that all DTopI$_{\text{reg}}$ have an equivalent leftmost earliest compatible DTopI$_{\text{reg}}$, we will prove that for all DTopI$_{\text{reg}}$, $can(\tau)$ is the unique minimal trimmed leftmost earliest compatible DTopI$_{\text{reg}}$, up to state renaming. To this end, we will prove that two equivalent leftmost DTopI$_{\text{reg}}$ have the same syntactically aligned pairs, which means they use equivalent states. Then, we will consider a minimal leftmost DTopI$_{\text{reg}}$ as a DTopI without two equivalent states, and prove that this is identical to $min(\tau)$ up to states renaming.

Lemma 120. *Le N a DTopI$_{reg}$, where $\tau = [\![N]\!]$, and $can(\tau) = (M, D)$, $M = (Q, F, G, ax, rhs)$. Then $can(\tau)$ is trimmed, and if $[p]_\tau$ and $[p']_\tau$ are two different states of Q, then $[\![can(\tau)]\!]_{[p]_\tau} \neq [\![can(\tau)]\!]_{[p']_\tau}$.*

Proof. By definition, no state of $can(\tau)$ is useless. Furthermore, a rule $rhs([p]_\tau, f)$ exists if and only if $f^{-1}dom(p^{-1}\tau) \neq \emptyset$. Since $[\![N]\!]_{[p]_\tau} = p^{-1}\tau$, this means that those rules are not useless. Hence, $can(\tau)$ is trimmed. Furthermore, if $[p]_\tau$ and $[p']_\tau$ are different states, then $p \not\equiv_\tau p'$, which means $p^{-1}\tau \neq p'^{-1}\tau$. Hence, $[\![can(\tau)]\!]_{[p]_\tau} \neq [\![can(\tau)]\!]_{[p']_\tau}$. □

We now prove that two leftmost transducers have very few possible differences. Notably, their syntactically aligned pairs are identical.

Theorem 121. *Let $N = (M, D)$ and $N' = (M', D)$ be two equivalent trimmed leftmost earliest compatible DTopI$_{reg}$. For all p syntactically aligned pair of N, p is a syntactically aligned pair of N'.*

Proof. Let $M = (Q, F, G, ax, rhs)$ and $M' = (Q', F, G, \{ax'\}, rhs')$. This proof is made by induction on the size of u. If $u = \varepsilon$ then for some state q of N, $ax \models vq\langle x_0\rangle$. Since M is earliest, Lemma 68 gives that $v^{-1}out_{[\![N]\!]}(\varepsilon) = \bot$. For the same reason, since $v^{-1}out_{[\![N']\!]}(\varepsilon) = \bot$, we have that for some q' of N', $ax' \models vq'\langle x_0\rangle$. Hence (u, v) if a syntactically aligned pair of N'.

If $u = u'fi$, then there exists v', v'' such that $v = v'v''$, $u' \sim_{q_0} v'$, and for some state q of N, $rhs(q_0, f) \models vq\langle x_i\rangle$. By induction, there exists a state q'_0 of N' such that $u' \sim_{q_0} v'$. This means that $[\![N]\!]_{q_0} = [\![N']\!]_{q'_0}$, and since N and N' are both trimmed and compatible, if $rhs(q_0, f)$ is defined, then $rhs'(q'_0, f)$ is defined. Since M is earliest, Lemma 68 gives that $v''^{-1}out_{[\![}N]\!]_{q_0}(f) = \bot$. For the same reason, since $v''^{-1}out_{[\![N']\!]_{q'_0}}(f) = \bot$, we have that for some q' of N', for some index j, $ax' \models vq'\langle x_j\rangle$. The fact that $i = j$ is due to the fact that both N and N' are both leftmost: $i = ind^{\text{left}}_\tau((u', v'), f, v'')$. □

We remind the definition of clear DTopI (Definition 88) and the fact that any earliest compatible DTopI has an equivalent clear earliest compatible DTopI (Lemma 89). Our aim is to prove that for every DTopI$_{\text{reg}}$ N there exists a *unique* clear leftmost earliest compatible DTopI$_{\text{reg}}$ equivalent to N.

Lemma 122. *For $N = (M, D)$ a trimmed leftmost earliest compatible DTopI$_{reg}$, and $N' = (M', D)$ an equivalent clear leftmost earliest compatible DTopI$_{reg}$. There exists an onto function ϕ from the states of N to the states of N' such that for all q state of N, $[\![N']\!]_{\phi(q)} = [\![N]\!]_q$.*

Proof. If q is a state of N, a trimmed DTopI, there exists (u, v) such that $u \sim_q v$. From Theorem 121 we conclude that there exists a state q' of N'

such that $u \sim_{q'} v$. For such a q', we would have $[\![N]\!]_q = [\![N']\!]_{q'}$. We note $\phi(q)$ the only state q' of the clear leftmost DTopI such that $[\![N]\!]_q = [\![N']\!]_{q'}$. This function is onto through a symmetrical reasoning: if q' is a state of N', a trimmed DTopI, there exists (u, v) such that $u \sim_{q'} v$. From Theorem 121 we conclude that there exists a state q of N such that $u \sim_q v$. For such a q, we would have $[\![N]\!]_q = [\![N']\!]_{q'}$, hence $\phi(q) = q'$. □

In the following theorem, we will prove that there only exists one clear leftmost DTopI$_{\text{reg}}$, up to state renaming, that is to say that if two DTopI$_{\text{reg}}$ N and N' are equivalent clear leftmost earliest compatible DTopI$_{\text{reg}}$, then there exists a one-to-one function ϕ from the states of N to the states of N' such that if every occurrence of every state q of N is replaced by its image $\phi(q)$, we obtain exactly N'.

Lemma 123. *If N and N' are two equivalent clear leftmost earliest compatible* DTopI$_{reg}$*, then $N = N'$, up to state renaming.*

Proof. From Lemma 122, we know that there exists a one-to-one correspondence ϕ between the states of N and the states of N' such that $[\![N]\!]_q = [\![N']\!]_{\phi(q)}$. We now show that this one-to-one correspondence is indeed a state rewriting between N and N'. For that, it just remains to prove that the rules of q and $\phi(q)$ are identical up to state renaming. First of all, since $[\![N]\!]_q = [\![N']\!]_{\phi(q)}$ and both N and N' are trimmed, we know that for all input letter f, there exist a rule $rhs(q, f)$ if and only if there is a tree of root f in $dom([\![N]\!]_q)$, if and only if there is a tree of root f in $dom([\![N']\!]_{\phi(q)})$, if and only if there exist a rule $rhs'(\phi(q), f)$. Furthermore, since both N and N' are earliest, from Lemma 68, we know that $rhs(q, f)\Psi = out_{[\![N]\!]_q}(f)$ and $rhs'(\phi(q), f)\Psi' = out_{[\![N']\!]_{\phi(q)}}(f)$ for some Ψ, Ψ'. Since $[\![N]\!]_q = [\![N']\!]_{\phi(q)}$, all that remains to show is that if $v^{-1}rhs(q, f) = q'\langle x_i\rangle$, then $v^{-1}rhs'(\phi(q), f) = \phi(q')\langle x_i\rangle$. Since $[\![N]\!]_q = [\![N']\!]_{\phi(q)}$, we have that if $v^{-1}rhs(q, f) = q'\langle x_i\rangle$, then $v^{-1}rhs'(\phi(q), f) = q''\langle x_j\rangle$. Since N is trimmed, there exists (u_0, v_0) such that $u_0 \sim_q v_0$ in N (and hence $u_0 \sim_{\phi(q)} v_0$ in N'). This means that for N, i is the the leftmost index for $(u_0 f, v_0 v)$, and $u_0 f i \sim_{q'} v_0 v$. Hence for N', i is the the leftmost index for $(u_0 f, v_0 v)$, and $u_0 f i \sim_{q''} v_0 v$. This means that if $v^{-1}rhs(q, f) = q'\langle x_i\rangle$, then $v^{-1}rhs'(\phi(q), f) = q''\langle x_j\rangle$, with $q'' = \phi(q')$ and $i = j$. □

This finally proves our normal form theorem:

Theorem 124. *For $N = (M, D)$ a* DTopI$_{reg}$*, there exists a unique equivalent leftmost compatible earliest* DTopI$_{reg}$ *with a minimal number of states, up to state renaming.*

Proof. The existence of such a DTopI is proven by Lemma 89, its uniqueness by Lemma 123. □

5.5 Learning from Examples

We next show how to learn DTopI$_{\text{reg}}$ under the condition that the domain is known a priori.

Definition 125. *Let F and G be ranked alphabet and $D \subseteq \mathcal{T}_F$ a regular language. We define* DTopI$_{reg}(D)$ *to be the class of transformations τ with regular domain $dom(\tau) = D$ and $\tau(D) \subseteq \mathcal{T}_G$ that are definable by some* DTopI$_{reg}$.

We extend Definition 94 to classes DTopI$_{\text{reg}}(D)$.

To extend the leaning algorithm of Chapter 4 to the more general case of DTopI$_{\text{reg}}(D)$ where D is a (non-empty) regular language, we must compensate where we previously relied on properties that held true on DTopI$_{\text{td}}$, but are not valid for all DTopI$_{\text{reg}}$. Three points are of particular interest. The first is that $Succ(p, f, v')$ is no longer necessarily unique, which means we have several possible successors to consider, even if we pick a sample with enough information to eliminate all non-functional candidates as described in point (3) of Definition 102. The second is that as seen in Example 113 some functional residuals can now not be defined by some DTopI. However, while learning if a residual is functional can easily be done from a finite sample (we suppose it is until proven otherwise), learning if a residual is definable by some DTopI is a more involved and complex process. The third difference is that Lemma 99 does not hold any more: in order to find a set of representative we can explore from the axiom successor by successor, $minp(\tau)$ is no longer a suitable definition.

5.5.1 Characteristic Samples

We start by addressing the problem of finding a suitable extension to $minp(\tau)$ as a set of representative that can be accessed from the axiom. We define $minexp(\tau)$ and $minexp^+(\tau)$ the set of minimally explored semantically aligned pairs. Our aim is to define a set that describes an exploration of aligned pairs successor by successor, that only keeps exploring the smallest pairs it encounters with new residuals:

Definition 126. *For any transformation τ we define the set of minimally explored semantically aligned pairs of τ $minexp(\tau)$ recursively as the smallest set such that:*

- *if $p = (\varepsilon, v) \in minp(\tau)$, then $p \in minexp(\tau)$*
- *if $p \in Succ(minexp(\tau))$, and there is no p' such that $p' \in minexp(\tau) \cup Succ(minexp(\tau))$, $p' < p$, $p' \equiv_\tau p$, then $p \in minexp(\tau)$*

For p a semantically aligned pair such that there exists $p' \in minexp(\tau)$, we note $minp_\tau(p) = p'$.

Definition 127. *For any transformation τ we define the successors of the minimally explored semantically aligned pairs of τ as:*

$$\begin{aligned} minexp^+(\tau) =& \{(\varepsilon, v) \mid v^{-1} out_\tau(\varepsilon) = \bot\} \cup \\ & \{p' \in Succ(minexp(\tau))\} \end{aligned}$$

It it worth noting that Lemma 99 proves that if $\tau \in \mathrm{DTopI}_{\mathrm{td}}$, $minp(\tau) = minexp(\tau)$, and $minp^+(\tau) = minexp^+(\tau)$. This is comforting, as $minexp(\tau)$ is supposed to be an extension of the notion of relevant pairs of τ to $\mathrm{DTopI}_{\mathrm{reg}}$, rather than a replacement of DTTA notion. We also note that, by construction, $minexp(\tau)$ has a property similar to Lemma 99, as any pair of $minexp(\tau)$ is either of form (ε, v) or a successor of another pair in $minexp(\tau)$.

Example 128. *We will present the example for τ_{ref}. We start from the axiom and then explore all new pairs according to the order we defined on pairs.*

- *Since $out_{\tau_{ref}}(\varepsilon) = \bot$, the pair $p_\varepsilon = (\varepsilon, \varepsilon)$ is in both $minexp(\tau_{ref})$ and $minexp^+(\tau_{ref})$.*

$$Succ_{\tau_{ref}}(p_\varepsilon) = \{(article1, \varepsilon), (article2, \varepsilon), (article3, \varepsilon)\}$$

  All three of these new pairs are in $minexp^+(\tau_{ref})$.
- *$p_1 = (article1, \varepsilon)$ is the smallest pair of its equivalence class we have encountered thus far, and is thus in $minexp(\tau_{ref})$. It has no semantic successor.*
- *$p_2 = (article2, \varepsilon)$ is the smallest pair of its equivalence class we have encountered thus far, and is thus in $minexp(\tau_{ref})$.*

$$Succ_{\tau_{ref}}(p_2) = \{(article2collection1, physical1), (article2website1, digital1), \}$$

  Both those new pairs are in $minexp^+(\tau_{ref})$.
- *$p_3 = (article3, \varepsilon)$ is the smallest pair of its equivalence class we have encountered thus far, and is thus in $minexp(\tau_{ref})$.*

$$Succ_{\tau_{ref}}(p_3) = \{(article3book1, physical1), (article3url1, digital1), \}$$

  Both those new pairs are in $minexp^+(\tau_{ref})$.
- *$p_A = (article2collection1, digital1)$ is the smallest pair of its equivalence class we have encountered thus far, and is thus in $minexp(\tau_{ref})$.*

$$Succ_{\tau_{ref}}(p_3) = \{(article3book1, physical1), (article3url1, digital1), \}$$

  It has no semantic successor.
- *All three remaining pairs have for residual $[a/a, a'/a']$ and are thus equivalent to p_A: they are not in $minexp(\tau_{ref})$.*
  - *$(article2website1, digital1)$*
  - *$(article3book1, physical1)$*

- $(article3url1, digital1)$

If $minexp(\tau)$ presents itself as a set of representatives that can be mapped to a subset of states in $can'(\tau)$, defining our learning target as was done in Definition 100 presents new challenges. Chief amongst them is the question of detecting pairs p whose residuals $p^{-1}\tau$ can be defined by some DTopI. This process has no known easy solution.

The solution proposed for this algorithm goes as follows: on the first step of the learning process, we create a *proto-transducer* $\widetilde{M} = (Q, F, G, ax, \widetilde{rhs})$, which differs from a classical transducer only in the fact that $\widetilde{rhs}$ is now a partial function from $Q \times F^{(k)}$ to $\mathcal{T}_G(2^{Q\times X_k})$. That is to say, each call in leaves of $\widetilde{rhs}(p, f)$ can now be a *set* (possibly empty) of calls to diverse states. The idea behind this new object is to describe all possible calls a valid rule could pick to form a proper DTop. Notably, if a rule $\widetilde{rhs}(q, f)$ contains an empty leaf $\emptyset$, it cannot be part of a proper DTop, which means its state q should be deleted. However, if a rule $\widetilde{rhs}(q, f)$ contains a leaf with more than two calls, we can pick any of these calls to form the valid rule of a proper DTop.

Our new learning target can therefore no longer be a DTopI, but a proto-transducer, that will be processed into a DTopI$_{\text{reg}}(D)$ in a second step after the learning process. We propose the following definition of $\widetilde{repcan}(\tau)$, derived from Definition 100:

Definition 129. *Let N a* DTopI$_{reg}(D)$, *and $\tau = [\![N]\!]$. We define $\widetilde{repcan}(\tau)$ as the proto-*DTop *$\widetilde{M} = (Q, F, G, ax, \widetilde{rhs})$, where:*
- *$Q = minexp(\tau)$*
- *$ax = out_\tau(\varepsilon)[v/minp_\tau(\varepsilon, v)\langle x_0\rangle \mid v^{-1}out_\tau(\varepsilon) = \bot]$*
- *For $p = (u, v) \in minp(\tau)$, f such that $uf^{-1}D \neq \emptyset$,*

$$\widetilde{rhs}(p, f) = out_{p^{-1}\tau}(f)[v'/\{minp_\tau(ufi, vv')\langle x_i\rangle \mid i \in Ind_\tau(p, f, v')\} \\ \mid v'^{-1}out_{p^{-1}\tau}(f) = \bot]$$

Example 130. *We give the proto-*DTop *$\widetilde{repcan}(\tau_{ref})$. We know from Example 128 that the pairs of $minexp(\tau_{ref})$ are $p_\varepsilon, p_1, p_2, p_3, p_A$. The axiom is $p_\varepsilon\langle x_0\rangle$. We now present the rules:*

$$(1)\ p_\varepsilon(article(x_1, x_2, x_3)) \to \{p_1\langle x_1\rangle, p_2\langle x_2\rangle, p_3\langle x_3\rangle\}$$

Here, three successors are available, thus, there are three calls.

$$(2)\ p_1(fullid(x_1, x_2, x_3)) \to \emptyset$$

p_1 has no semantic successors, hence the empty call set.

$$\begin{array}{ll} (3) & p_2(website(x_1)) \to digital(\{p_A\langle x_1\rangle\}) \\ (4) & p_2(collection(x_1)) \to physical(\{p_A\langle x_1\rangle\}) \\ (5) & p_3(url(x_1, x_2)) \to digital(\{p_A\langle x_1\rangle\}) \\ (6) & p_3(book(x_1, x_2)) \to physical(\{p_A\langle x_1\rangle\}) \end{array}$$

Note that all calls go to p_A. For the example of rule (6), $Succ_{\tau_{ref}}(p_3, book, physical1) = \{(article3book1, physical1)\}$. *Since p_A is the pair equivalent to $(article3book1, physical1)$ in $minexp(\tau_{ref})$, the call in rule* (6) *is* $p_A\langle x_1\rangle$

$$(9)\quad p_A(a) \to a \qquad (10)\quad p_A(a') \to a'$$

Note that since rule (2) *has an empty call set, state p_1 is improper and will be deleted when building the normal form. Then, rule* (1) *can choose $p_2\langle x_2\rangle$ as its leftmost call, leaving p_3 inaccessible from the axiom. After this reasoning, we would build from $\widetilde{repcan}(\tau_{ref})$ a minimal leftmost earliest compatible* $\mathrm{DT\textsc{op}I}_{reg}$ *for τ_{ref}.*

We now extend the definition of characteristic to the $\mathrm{DT\textsc{op}I}_{\mathrm{reg}}$ case. A sample now needs enough information to build $\widetilde{repcan}(\tau)$. The main differences to adapt to the $\mathrm{DT\textsc{op}I}_{\mathrm{reg}}$ case are that S needs information on *minexp* rather than *minp*, and Point (3) needs to be reformulated to accommodate with the fact that more than one successor can exist for the same output path.

Definition 131. *Let N be a* $\mathrm{DT\textsc{op}I}_{reg}(D)$ *in normal form and $\tau = [\![N]\!]$. A sample S for τ is called* characteristic *if:*

(1) $out_S(\varepsilon) = out_\tau(\varepsilon)$

(2) for all $p \in minexp(\tau)$ and $f \in F$: $out_{p^{-1}S}(f) = out_{p^{-1}\tau}(f)$,

(3) for $p = (u, v) \in minexp(\tau)$, $f^{(k)} \in F$, and v' such that $v'^{-1}out_{p^{-1}\tau}(f)$, for all $1 \leq j \leq k$, such that $(ufj, vv')^{-1}\tau$ is not functional, then $(ufj, vv')^{-1}S$ is not functional.

(4) for all $p \in minexp^+(\tau)$, $p' \in minexp(\tau)$ such that $p^{-1}D = p'^{-1}D$, and $p \not\equiv_\tau p'$: $p \not\Vert_S p'$.

We now establish that such a sample always exists in a number of examples polynomial in the size of $minexp^+(\tau)$.

Proposition 132. *Let τ be a transformation definable by a* $\mathrm{DT\textsc{op}I}_{reg}(D)$. *Then there exists a sample S characteristic for τ with a number of examples polynomial in the size of $minexp^+(\tau)$.*

Proof. We will show that a polynomial number of examples is required for each point (1-4) of Definition 102.

For point (1), we want to ensure that $out_S(\varepsilon) = out_\tau(\varepsilon)$. First, we need at least one example in to ensure that $out_S(\varepsilon)$ is defined. Note that such an example always exists, as we supposed $dom(\tau) = D \neq \emptyset$. We fix $s_\varepsilon \in dom(\tau)$ arbitrarily and add $(s_\varepsilon, \tau(s_\varepsilon))$ to S. Then, since $S \subseteq \tau$ will be guaranteed, the only concern is that $out_S(\varepsilon)$ is bigger than $out_\tau(\varepsilon)$. To this end, we will provide one example per equivalence class $[p]_\tau$ of $\equiv_\tau$, where $p = (\varepsilon, v)$ is an aligned pair of τ. For each such class $[p]_\tau$, there exists a tree $s_{[p]_\tau}$ such that $v^{-1}\tau(s_{[p]_\tau}) \sqcap v^{-1}\tau(s_\varepsilon) = \bot$. If $(s_\varepsilon, \tau(s_\varepsilon)) \in S$ and for all equivalence class

$[p]_\tau$ of $\equiv_\tau$, where $p = (\varepsilon, v)$ is an aligned pair of τ, $(s_{[p]_\tau}, \tau(s_{[p]_\tau})) \in S$, then $out_S(\varepsilon) = out_\tau(\varepsilon)$.

Point (2) works in a similar fashion. We want to ensure that for all $p = (u, v) \in minexp(\tau)$, for all input letter f, $out_{p^{-1}S}(f) = out_{p^{-1}\tau}(f)$. First, we need at least one example such that $s_{p,f} \models uf$ to ensure that $out_{p^{-1}S}(f)$ is defined. We call this example $(s_{p,f}, \tau(s_{p,f}))$. Then, since $S \subseteq \tau$, the only concern is that $out_{p^{-1}S}(f)$ is bigger than $out_{p^{-1}\tau}(f)$. We shall ensure that there exists enough examples in S so that for all v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \bot$, $v'^{-1}out_{p^{-1}S}(f) = \bot$. To this end, we will provide one example per equivalence class $[p']_\tau$ of $\equiv_\tau$, where $p' \in Succ_\tau(p)$. For each such class $[p']_\tau$, there exists a tree s such that $p'^{-1}\tau(s) \sqcap p'^{-1}\tau(v^{-1}s_{p,f}) = \bot$. We then choose a tree $s_{p,f,[p']_\tau}$, such that $u^{-1}s_{p,f,[p']_\tau} = s$. If $(s_{p,f}, \tau(s_{p,f})) \in S$ and for all $v'^{-1}out_{p^{-1}\tau}(f) = \bot$, $p' \in Succ_\tau(p)$, $(s_{p,f,[p']_\tau}, \tau(s_{p,f,[p']_\tau})) \in S$, then $out_{p^{-1}S}(f) = out_{p^{-1}\tau}(f)$.

Point (3) is ensured by providing an explicit counterexample for every pair $p' = (ufj, vv')$ we want to prove is not functional. If $p'^{-1}\tau$ is not functional, then there exists $s_{p'}$, $s'_{p'}$ input trees such that $ufj^{-1}s_{p'} = ufj^{-1}s'_{p'}$ but either $vv'^{-1}\tau(s_{p'}) \neq= vv'^{-1}\tau(s'_{p'})$ or $\tau(s_{p'}) \models vv'$ but $\tau(s'_{p'}) \not\models vv'$. Hence, if S contains $(s_{p'}, \tau(s_{p'}))$ and $(s'_{p'}, \tau(s'_{p'}))$, then $p'^{-1}S$ is not functional. Note that there is a polynomial number of those pairs: for $p = (u, v) \in minexp(\tau)$, for f an input letter, all paths v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \bot$ are in $rhs(q, f)$ where q is the state of N that computes $p^{-1}\tau$. Since $j \leq rank(f)$, this leaves a polynomial number of pairs to consider.

Point (4) works in a similar fashion. For $p = (u, v)$, $p' = (u', v')$ two non-equivalent semantic alignments, if $p^{-1}D = p'^{-1}D$, then there exists s an input tree such that $p{-}1\tau(s) \neq p'{-}1\tau(s)$. We take two input trees $s_{p,p'}$, $s'_{p,p'}$ such that $u^{-1}s_{p,p'} = u'^{-1}s'_{p,p'} = s$. Hence, if S contains $(s_{p,p'}, \tau(s_{p,p'}))$ and $(s'_{p,p'}, \tau(s'_{p,p'}))$, then $p'^{-1}S$ is not functional. Note that the number of cases to consider is polynomial in the size of $minexp^+(\tau)$.

By taking all examples needed to ensure points (1-4), we built a characteristic sample for τ polynomial in the size of $minexp^+(\tau)$. □

The goal of this part is to prove that one can learn a DTopI$_{\text{reg}}(D)$ with a finite number of examples. Contrary to the DTopI$_{\text{td}}(D)$ case, the number of examples is not polynomial in the size of the target normal form, but in the number of representatives in $minexp^+(\tau)$.

Theorem 133. *For any regular domain D, the class* DTopI$_{reg}(D)$ *is learnable.*

Proof. The proof captures the rest of this section. We will prove that a learning algorithm $learn_D$ exists, of polynomial complexity. Proposition 132 shows that the number of examples this algorithm needs is polynomial in the number of representatives in $minexp^+(\tau)$. □

5.5.2 Learning Algorithm

We want to adapt the algorithm of Figure 4.5 to learn the proto-transducer $\widetilde{repcan}(\tau)$ instead of directly learning a renaming of the normal form of Theorem 124. We aim to explore pairs of *minexp*, successor by successor, to build a proto-transducer in the same way $learn_D$ builds a DTOP in the DTOPI_{td} case.

However, the $\text{DTOPI}_{\text{reg}}$ case requires an additional step. We have to transform the proto-transducer we learnt into a proper $\text{DTOPI}_{\text{reg}}(D)$. If at some point a rule $\widetilde{rhs}(p, f)$ has a leaf with an empty set, then it has no functional successors: its residual is not computed with a $\text{DTOPI}_{\text{reg}}$, and p should be deleted from the states. If in the end several choices are possible, we pick the leftmost choice, in order to learn the canonical normal form described in Section 5.4. Finally, we trim the resulting transducer from unaccessible states.

In the first function $learn_D$, we build a proto-transducer $learn_D(S) = (Q, F, G, ax, \widetilde{rhs})$, where for simplicity's sake, our states will be pair (u, v). Those states will be divided in two disjoint sets: Q_{safe} for pairs that minimally represent an equivalence class, and therefore represent a state in $learn_D(S)$, and Q_{temp}, for pairs that have not yet been examined by the algorithm, and are still susceptible to be equivalent to an existing pair in Q_{safe}. Rules of $\widetilde{rhs}$ are only created for states of Q_{safe}, but leaves in these rules or the axiom can now be *sets* of pairs $p\langle x_i\rangle$, where p is either a safe state of Q_{safe}, or a yet unapproved pair of Q_{temp}.

$learn_D$ is almost identical to the DTOPI_{td} case: its only changes concern the new set nature of $\widetilde{rhs}$ leaves. It will then require an additional "refining" algorithm to output a proper transducer. The creation of the axiom is unchanged. *integrate-state* stays relatively unchanged too, but accommodates for the possibility of having 0 or several successors in the leaves of $\widetilde{rhs}$.

To the first part of the algorithm, quite similar to the DTOPI_{td} case, we add three post-treatment of the produced transducer. First, we want to delete pairs that fail to compute their residuals (a rule has $\emptyset$). Then, if in some rules some choice remains, we select the leftmost choice. Finally, in order to ensure we have the canonical transducer for τ, we trim the transducer by deleting all states non-accessible from the axiom.

To prove this learning algorithm, we proceed in two step: the first one is to prove that $learn_D$ produces $\widetilde{repcan}(\tau)$. As was the case in the DTOPI_{td} case, this is shown using a loop invariant on $learn_D$, as its variable $\widetilde{M}$ is always a *p-truncated* proto-transducer, with a definition quite similar to Definition 107. Once we ensure that $learn_D$ produces $\widetilde{repcan}(\tau)$ on a characteristic sample, we prove that *cull* transforms $\widetilde{repcan}(\tau)$ into a minimal leftmost earliest compatible $\text{DTOPI}_{\text{reg}}(D)$ defining τ.

We introduce the *p-truncated proto-transducer*, $\widetilde{repcan}_p(\tau)$ that represents

```
// let $F$ and $G$ be ranked signatures
// and $D \subseteq \mathcal{T}_F$ the language of some tree automaton

fun $learn_D(S)$ // $S \subseteq D \times \mathcal{T}_G$ finite sample
  $Q_{temp}$ := $\{(\varepsilon, v) \mid v^{-1} out_S(\varepsilon) = \bot\}$
  $ax$ = $out_S(\varepsilon)[v/(\varepsilon,v)\langle x_0\rangle \mid (\varepsilon,v) \in Q_{temp}]$
  $Q_{safe}$ := $\emptyset$
  $\widetilde{rhs}$ := $\emptyset$
  $\widetilde{M}$ = $(Q_{safe} \cup Q_{temp}, F, G, ax, \widetilde{rhs})$
  proc $integrate\text{-}state(p)$ = // either fusion temporary
        // state $p$ with some safe state or make $p$
        // safe and add its transition rules.
    $Q_{eq}$ = $\{p' \in Q_{safe} \mid p^{-1}D = p'^{-1}D$ and not $p \not\Vdash_S p'\}$
    $(u, v) = p$
  in
    case $Q_{eq}$ // $Q_{eq}$ may contain at most 1 element
    of $\{p'\}$ then replace all occurrences $of\ p\ in\ M$ by $p'$
    of $\emptyset$ then
       $Q_{safe}$ := $Q_{safe} \cup \{p\}$
       $Q_{temp}$ := $Q_{temp} \setminus \{p\}$
       for $f \in F$ where $uf^{-1}D \neq \emptyset$ do
         $V' = \{v' \mid v'^{-1} out_{p^{-1}S}(f) = \bot\}$
         fun $I(v')$ // where $v' \in V'$
           $\{i \mid (ufi, vv')^{-1}S$ is functional$\}$
         end
         fun $states(v')$ // where $v' \in V'$
           $\{(ufi, vv') \mid i \in I(v')\}$
         end
         fun $calls(v')$ // where $v' \in V'$
           $\{(ufi, vv')\langle x_i\rangle \mid i \in I(v')\}$
         end
           $o$ = $out_{p^{-1}S}(f)[v'/calls(v') \mid v' \in V']$
       in
         $Q_{temp}$ := $Q_{temp} \cup \left(\bigcup_{v' \in V'} states\,(v')\right)$
         $rhs(p, f)$ := o
       end
    end
  end
in
   while $Q_{temp} \neq \emptyset$ do
     $p$ = $min(Q_{temp})$
   in
     $integrate\text{-}state(p)$
   end
   return $\widetilde{M}$
end
```

Figure 5.2: Learning algorithm for DTops with regular inspection.

```
// let F and G be ranked signatures
fun $cull(\widetilde{M})$ // $\widetilde{M}$ a proto−transducer
  $\widetilde{M} = (Q, F, G, ax, \widetilde{rhs})$
  $Q_{dead}$ := $\{p \in Q \mid \exists f, v \mid v^{-1}\widetilde{rhs}(p, f) = \bot\}$

  // Cull dead−end states
  while $Q_{dead} \neq \emptyset$ do
    for $p$ where $p \in Q_{dead}$ do
      Delete all rules $\widetilde{rhs}(p, f)$
      Delete all occurrences of $p$ in $\widetilde{rhs}$
    end
    $Q_{dead}$ := $\{p \in Q \mid \exists f, v \mid v^{-1}\widetilde{rhs}(p, f) = \bot\}$
  end

  // Make leftmost choice
  $rhs$ := $\emptyset$ // set of real rules
  for $p, f$ where $\widetilde{rhs}(p, f)$ defined
    fun $call(v)$ // for $v$ where $v^{-1}\widetilde{rhs}(p, f) \in 2^{Q\times X}$
      $p'\langle x_i\rangle$ such that $p'\langle x_i\rangle \in v^{-1}\widetilde{rhs}(p, f)$
              and $\nexists p'', j$ such that $j < i$ and $p''\langle x_j\rangle \in v^{-1}\widetilde{rhs}(p, f)$
    end
    $rhs(p, f)$ := $\widetilde{rhs}(p, f)[v/call(v) \mid v^{-1}\widetilde{rhs}(p, f) \in 2^{Q\times X}]$
  end


  // Trim inaccessible states
  $Q_{old}$ := $\emptyset$
  $Q_{new}$ := $\{p \mid p$ occurs in $ax\}$
  while $Q_{old} \neq Q_{new}$ do
    $Q_{old}$ := $Q_{new}$
    $Q_{new}$ := $\{p \mid p$ occurs in $rhs(Q_{old} \times F)\}$
  end
  Delete all rules $rhs(Q \setminus Q_{new} \times F)$
  $Q$ := $Q_{new}$

  $M$ = $(Q, F, G, ax, rhs)$
  return $(M, D)$
end
```

Figure 5.3: Deletes all dead-end states in a proto-transducer, then make a "left-most" choice to make it a transducer, then trim it.

the way the proto-transducer built in $learn_D$ should look right before calling $integrate\text{-}state(p)$ if everything went right. This definition is an adaptation of Definition 107 to describe a partially folded proto-transducer $\widetilde{repcan}_p(\tau)$ rather than a partially folded transducer $repcan_p(\tau)$.

Definition 134. *Let N be a* $\mathrm{DT{\small OP}I}_{reg}(D)$*, $\tau = [\![N]\!]$, and p an aligned pair of τ. For $p' \in minexp^+(\tau)$, we call $minrep^p_\tau(p')$ the p-truncated representative of p':*

- *$minrep^p_\tau(p') = p''$ the unique element of $minexp(\tau)$ such that $p'' \equiv_\tau p'$ if $p' < p$,*
- *$minrep^p_\tau(p') = p'$ itself if $p' \geqslant p$*

Definition 135. *Let N be a* $\mathrm{DT{\small OP}I}_{reg}(D)$*, $\tau = [\![N]\!]$, $\widetilde{repcan}(\tau) = (Q, F, G, ax, \widetilde{rhs})$, and p an aligned pair of τ. We define the p-truncated form of $\widetilde{repcan}(\tau)$, $\widetilde{repcan}_p(\tau) = (Q_{safe(p)} \cup Q_{temp(p)}, F, G, ax_p, \widetilde{rhs}_p)$, where:*

- *The p-truncated safe states $Q_{safe(p)} = \{p' \in Q \mid p' < p\}$*
- *The p-truncated temporary states*
  $Q_{temp(p)} = \big(\{(\varepsilon, v) \mid v^{-1}ax = q\langle x_0\rangle\} \cup Succ_\tau(Q_{safe(p)})\big) \cap \{p' \mid p' \geqslant p\}$
- *The $p-truncated$ axiom*
  $ax_p = ax\ [v/minrep^p_\tau(\varepsilon, v)\langle x_0\rangle \mid v^{-1}ax = q\langle x_0\rangle]$
- *The p-truncated proto-rules $\widetilde{rhs}_p$ are defined so that if $p' = (u', v') \in Q_{safe(p)}$, and f a letter such that $\widetilde{rhs}(p', f)$ is defined, then*
  $\widetilde{rhs}_p(p', f) = \widetilde{rhs}(p', f)\Phi$*, where*

$$\Phi = [v''/\{minrep^p_\tau(u'fi, v'v'')\langle x_i\rangle \mid \exists p'\langle x_i\rangle \in v''^{-1}rhs(p', f)\} \\ \mid v''^{-1}rhs(p', f) \in 2^{Q\times X}]$$

Note that as in the case of p-truncated transducers, this definition "peaks" for $p > max(minexp^+(\tau))$, by reaching $\widetilde{repcan}(\tau)$.

Corollary 136. *Let N be a* $\mathrm{DT{\small OP}I}_{reg}(D)$*, $\tau = [\![N]\!]$. For all semantically aligned pair p such that $p > max(minexp^+(\tau))$, $\widetilde{repcan}_p(\tau) = \widetilde{repcan}(\tau)$*

Proof. By construction, if $p > max(minexp^+(\tau))$, for all pairs $p' \in minexp^+(\tau)$, $minrep^p_\tau(p') = minp_\tau(p')$. Furthermore, $Q_{safe(p)} = minexp(\tau)$, $Q_{temp(p)} = \emptyset$ hence $\widetilde{repcan}_p(\tau)$ and $\widetilde{repcan}(\tau)$ have same states. Finally, the only difference between the definition of the axiom and proto-rules of $\widetilde{repcan}_p(\tau)$ and $\widetilde{repcan}(\tau)$ is that the former uses $minrep^p_\tau$ and the latter uses $minp_\tau$. Since they are both identical for all pairs of $minexp^+(\tau)$, we have that $\widetilde{repcan}_p(\tau)$ and $\widetilde{repcan}(\tau)$ have same axioms and proto-rules. □

We prove that $learn_D$ on S a characteristic sample for τ produces $\widetilde{repcan}(\tau)$. To this end, we show that right before each call to $integrate\text{-}state(p)$, the proto-transducer $\widetilde{M}$ created by $learn_D$ is exactly $\widetilde{M}_p$.

Theorem 137. *Let N be a* $\mathrm{DT}\textsc{op}\mathrm{I}_{reg}(D)$, $\tau = [\![N]\!]$, $\widetilde{M_\tau}$ *the canonical proto-transducer of* τ, S *a characteristic sample for* τ. *Then* $learn_D(S) = \widetilde{repcan}(\tau)$.

Proof. As in the $\mathrm{DT}\textsc{op}\mathrm{I}_{\mathrm{td}}$ case, we prove the following invariant: if $Q_{temp} \neq \emptyset$ and $p = min(Q_{temp})$, then $Q_{safe} = Q_{safe(p)}$, $Q_{temp} = Q_{temp(p)}$, $ax = ax_p$ and $\widetilde{rhs} = \widetilde{rhs}_p$.

After the initialization, $Q_{safe} = \emptyset$ and $Q_{temp} = \{(\varepsilon, v) \mid v^{-1}out_S(\varepsilon) = \bot\}$. Since S is characteristic, this means $Q_{temp} = \{(\varepsilon, v) \mid v^{-1}out_\tau(\varepsilon) = \bot\}$. If $Q_{temp} \neq \varepsilon$, we call $p = min(Q_{temp})$. p is the smallest aligned pair of τ. This means that $Q_{safe(p)} = minp \cap \{p' \mid p' < p\}$ is empty, and therefore $Q_{safe(p)} = Q_{safe}$. This gives us that $Q_{temp(p)} = \{(\varepsilon, v) \mid v^{-1}out_\tau(\varepsilon) = \bot\} \cap \{p' \mid p' \geqslant p\}$. Since all aligned pairs are bigger than p, we have $Q_{temp(p)} = Q_{temp}$.
Since p is the smallest aligned pair, $minrep^p_\tau$ is the identity function. This means that $ax_p = out_\tau(\varepsilon)\ [v/(\varepsilon, v)\langle x_0\rangle \mid v^{-1}out_\tau(\varepsilon) = \bot]$. The current axiom in learn is $ax = out_S(\varepsilon)\ [v/(\varepsilon, v)\langle x_0\rangle \mid v^{-1}out_S(\varepsilon) = \bot]$. Since S is characteristic, $ax_p = ax$. As for rules, none have been created yet, which means $\widetilde{rhs} = \widetilde{rhs}_p = \emptyset$.

For the inductive case, we consider $p = (u, v)$ the minimal element of Q_{temp}. We have $Q_{safe} = Q_{safe(p)}$, $Q_{temp} = Q_{temp(p)}$, $ax = ax_p$ and $\widetilde{rhs} = \widetilde{rhs}_p$. We call the new values after *integrate-state*(p) Q'_{safe}, Q'_{temp}, ax' and $\widetilde{rhs}'$. If $Q'_{temp} \neq \emptyset$, we call p' its minimum. p is added to Q_{safe} if and only if for all $p'' \in Q_{safe}$, $p \not\parallel_S p''$. Since S is characteristic, $Q_{safe} \subseteq minp(\tau)$ and $p \in minp^+(\tau)$, this means that for all $p'' \in Q_{safe(p)}$, $p \not\equiv_\tau p''$. Hence, p is added to Q_{safe} if and only if $p \in minp(\tau)$. For the same reason, states are added to Q_{temp} if and only if $p \in minp(\tau)$. If S is characteristic, then $V' = \{v' \mid v'^{-1}out_\tau(f) = \bot\}$, and for each $v' \in V'$, then $I(v')$ is the set of all i such that $(ufi, vv')^{-1}\tau$ is functional, i.e. $I(v') = Ind_\tau(p, f, v')$. This means that the new temporary states of Q'_{temp} are $\{state(v') \mid v' \in V'\} = Succ_\tau(p)$. In all cases p is removed from Q_{temp}. This means that regardless of weather p was added or not, $Q'_{safe} = minp(\tau) \cap \{p'' \mid p'' \leqslant p\}$ and $Q'_{temp} = \{(\varepsilon, v) \mid v^{-1}out_\tau(\varepsilon) = \bot\} \cap \{p'' \mid p'' > p\}$. This means that Q'_{safe} and Q'_{temp} are $Q_{safe(p'')}$ and $Q_{temp(p'')}$ for some pair p'' right after p. Since $p' = min(Q_{temp})$, there is no element of $minp^+(\tau)$ between p and p'. Thus, $Q'_{safe} = Q_{safe(p')}$, and $Q'_{temp} = Q_{temp(p')}$.
If p is not a new state, then $minrep^p_\tau$ is different from $minrep^{p'}_\tau$ only for p, which has to be replaced by the only $p'' \in Q_{safe}$ such that $p'' \equiv_\tau p$. As *integrate-state*(p) replaces every state call $p\langle x_i\rangle$ by $p''\langle x_i\rangle$, we have $ax' = ax_{p'}$ and $\widetilde{rhs}' = \widetilde{rhs}_{p'}$. However, if p is a new state, then $minrep^{p'}_\tau = minrep^p_\tau$, and thus $ax' = ax_{p'}$, but new proto-rules have to be added. *integrate-state*(p) adds new proto-rules. For $p = (u, v)$, we have that for each f such that $uf^{-1}D \neq \emptyset$, we create the rule $\widetilde{rhs}'(p, f) = out_{p^{-1}S}(f)\ [v'/calls(v') \mid v' \in V']$. As previously mentioned, $V' = \{v' \mid v'^{-1}out_\tau(f) = \bot\}$, and $I(v') = Ind_\tau(p, f, v')$. Hence $\widetilde{rhs}'(p, f) = out_{p^{-1}\tau}(f)\ [v'/\{(ufi, vv')\langle x_i\rangle \mid i \in Ind_\tau(p, f, v')\}, v' \in V']$.

For all (ufi, vv') in $Succ_\tau(p)$, we know that $(ufi, vv') > p$, and thus $p'' = minrep_\tau^{p'}(p'')$. This gives $\widetilde{rhs}' = \widetilde{rhs}_{p'}$.

It remains to show that the last step that eventually empties Q_{temp} leads to $\widetilde{repcan}(\tau)$. If Q_{temp} starts as $\emptyset$, this means $out_S(\varepsilon)$ has no $\bot$-leaf. Since S is characteristic, this means $out_\tau(\varepsilon)$ has no $\bot$-leaf. This is only possible if τ is a constant transduction that sends all trees of D to the same image t. In this case, $learn_D(S)$ produces a transducer with no states and no rules, and an axiom $ax = t$, which is indeed $\widetilde{repcan}(\tau)$ (and even $repcan(\tau)$). In all other cases, we consider p the last pair to be integrated by $integrate\text{-}state(p)$. Since it is the last considered pair, we have that $p = max(minp^+(\tau))$. As seen in this proof, after this last $integrate\text{-}state(p)$, we have $learn_D(S) = \widetilde{repcan}_{p'}(\tau)$, where p' is the pair right after p in lexical order. Thus, Corollary 136 gives us that $learn_D(S) = repcan(\tau)$. □

From there, proving the correctness of our algorithm hinges on one last step: proving that *cull* transforms $\widetilde{repcan}(\tau)$ into a minimal leftmost earliest compatible DTopI$_{\text{reg}}(D)$.

Proposition 138. *Let N a* DTopI$_{reg}(D)$*, and $\tau = [\![N]\!]$. Then $cull(\widetilde{can}(\tau))$ is a minimal leftmost earliest compatible* DTopI$_{reg}(D)$ *N' such that $[\![N']\!] = \tau$.*

Proof. This proof works in three steps. The first step proves by induction that all states p deleted during the first loop of *cull* are such that $p^{-1}\tau$ cannot be defined by a DTopI. The second step is reminiscent of Proposition 119, and shows that for $N' = (M, D)$ the transducer produced after the leftmost choice, $[\![N']\!] = \tau$, and for each state p, $[\![N']\!]_p = p^{-1}\tau$. The third step aims to use Theorem 124, and shows that $cull(\widetilde{repcan}(\tau))$ is a minimal leftmost earliest compatible DTopI$_{\text{reg}}(D)$ that defines τ, i.e. $can(\tau)$, up to state renaming.

First, we show that all states p deleted during the first loop of *cull* are such that $p^{-1}\tau$ cannot be computed by a DTop. This can be done by recursion on the loop. If a state $p = (u, v)$ is deleted, this means that $p \in Q_{dead}$, i.e. there is a letter f and a path v' such that $v^{-1}\widetilde{rhs}(p, f) = \emptyset$. From Definition 129, we know that originally, $v'{-}1\widetilde{rhs}(p, f) = calls(v')$, i.e. the set $\{minrep(ufi, vv')\langle x_i\rangle \mid i \in Ind(p, f, v')\}$. If $v'^{-1}\widetilde{rhs}(p, f) = \emptyset$, this means that all states $\{minrep(ufi, vv') \mid i \in Ind(p, f, v')\}$ were deleted. This means that for all i, $(ufi, vv')^{-1}\tau$ is either not functional, or $minrep(ufi, vv')$ was deleted, which by recursion means $(ufi, vv')^{-1}\tau$ cannot be computed by a DTopI. This means that there exists a v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \bot$, but there is no i such that $(fi, v')^{-1}(p^{-1}\tau)$ can be described by a DTopI. This means that $p^{-1}\tau$ itself cannot be described by a DTopI.

Second, we show that after the leftmost choice, the DTopI$_{\text{reg}}(D)$ $N' = (M, D)$ where $M = (Q, F, G, ax, rhs)$ defines τ, and for each state $p \in Q$,

$[\![N']\!]_p = p^{-1}\tau$. This proof is similar to Proposition 119: we prove it by induction on the input tree. For a tree $s = f(s_1, ..., s_n)$ we have that

$$[\![N']\!]_p(s) = out_{p^{-1}\tau}(f)[v'/[\![N']\!]_{minrep(ufi,vv')}(s_i) \mid v'^{-1}rhs(p,f) = minrep(ufi,vv')\langle x_i\rangle]$$

By induction, $[\![N']\!]_{minrep(ufi,vv')}(s_i) = (ufi, vv')^{-1}\tau(s_i) = v'^{-1}p^{-1}\tau(s)$. We can then prove $[\![N']\!]_p(s) = p^{-1}\tau(s)$. All paths v'' such that $out_{p^{-1}\tau}(f) \models v''$ are both in $p^{-1}\tau(s)$ and in $[\![N']\!]_p(s)$. Plus, for v' such that $v'^{-1}out_{p^{-1}\tau}(f) = \perp$, $v'^{-1}[\![N']\!]_p(s) = v'^{-1}p^{-1}\tau(s)$. We add the axiom of $\widetilde{repcan}(\tau)$ on top of these production to show that $\tau = [\![N']\!]$. For a tree $s \in dom(\tau)$ we have that $[\![N']\!](s) = out_\tau(\varepsilon)[v/[\![N']\!]_{minrep(\varepsilon,v)}(s) \mid v^{-1}out_\tau(\varepsilon) = \perp]$ We can then prove $[\![N']\!](s) = \tau(s)$. All paths v' such that $out_\tau(\varepsilon) \models v'$ are both in $\tau(s)$ and in $[\![N']\!](s)$. Plus, for v such that $v^{-1}out_\tau(\varepsilon) = \perp$, $v^{-1}[\![N']\!](s) = v^{-1}\tau(s)$.

The third step ensures that after the deletion of inaccessible states, N' is a minimal leftmost earliest compatible DTopI$_{\text{reg}}(D)$. To show that N' is compatible, we consider that if p_1 and p_2 are two pairs syntactically aligned in the same state p, then $p_1^{-1}\tau = p_2^{-1}\tau = p^{-1}\tau$. To show that N' is earliest, we consider that if p' is syntactically aligned in the state p, then $p'^{-1}\tau = p^{-1}\tau$. Since by definition of $\widetilde{repcan}(\tau)$, p is semantically aligned, this means that p' is semantically aligned. To show that N' is leftmost, we recall the first step of this proof: if $v'^{-1}rhs(p,f) = minrep(ufi, vv')\langle x_i\rangle$, this means that i is the minimal index such that $minrep(ufi, vv')$ was not deleted in the first loop. This means that for all $j < i$, $(ufj, vv')^{-1}\tau$ cannot be defined by a DTopI. Hence, N' is leftmost. Finally, to show that N' is minimal, we note that if p and p' are two different states of Q, then $p \not\equiv_\tau p'$. Therefore, there is no redundant states. Furthermore, since $[\![N']\!]_p = p^{-1}\tau$, if p is accessible from the axiom, then it is useful. Since N' only keeps states accessible from the axiom, it is trimmed.

Hence, N' is a minimal leftmost earliest compatible DTopI$_{\text{reg}}(D)$ that defines τ, i.e. $can(\tau)$, up to state renaming. □

Chapter 6
Learning Rational Functions

Abstract. Rational functions are transformations from words to words that can be defined by nondeterministic string transducers. Rational functions can also be captured by deterministic string transducers with lookahead. We show for the first time that the class of rational functions can be learned in the limit with polynomial time and data, when represented by string transducers with lookahead in the diagonal-minimal normal form that we introduce.

Contents

6.1 Introduction 126
6.2 Rational Functions 126
6.3 Transducers with Look-Ahead 128
6.4 Building the Look-Ahead Automaton 131
6.5 The Learning Algorithm 133
6.6 Characteristic Samples 134

6.1 Introduction

In this chapter, we aim to study the learning problem for rational functions, i.e. functions that can be described by a nondeterministic word transducer. This learning problem – that remained open for many years – is whether one can learn rational functions from finite samples of input-output examples and a DFA for the domain.

Previous works have shown that subsequential tree transducers (or DWTs) can be learnt in Gold's learning model in polynomial time from a polynomial number of examples (OSTIA algorithm [Oncina and Varo, 1996]). We wish to extend that result to the wider class of deterministic word transducers with lookahead (DWT$_\ell$), which capture the class of rational functions (see e.g. [Berstel, 1979]), i.e. they have the same expressiveness as functional nondeterministic word transducers [Elgot and Mezei, 1965]. Based on another Myhill-Nerode type theorem, Reutenauer and Schützenberger showed in [Reutenauer and Schützenberger, 1991] that there exists a unique minimal look-ahead automaton compatible with the domain that can be used to define some DWT$_\ell$. The underlying DWT itself can be made earliest and minimal. This yields a unique two-phase minimal normal form for rational functions.

We provide a learning algorithm in Gold's learning model in polynomial time from a polynomial number of examples, under the assumption that rational functions are represented by diagonal-minimal normal form. This is a new class of normal forms that we introduce concommitantly with a new learning algorithm based on diagonalization. The main problem was to overcome the difficulty to identify a two-phase minimal normal form from examples.

Outline. We first recall traditional results on rational and subsequential functions (Section 6.2) and then the result of Reutenauer and Schützenberger on two-phase DWT$_\ell$ normalization (Section 6.3). In section 6.4, we indicate how to build a look-ahead from a basic test over suffixes. In Section 6.6, we indicate how this test can be done from a finite sample which leads to section 6.5 where we present the complete learning algorithm.

6.2 Rational Functions

We assume an input alphabet Σ and an output alphabet Δ, both of which are finite sets. Input words in Σ^* are ranged over by u and v and output words in Δ^* by w. We are interested in partial functions $\tau \subseteq \Sigma^* \times \Delta^*$. We denote the domain of a partial function by $dom(\tau)$ and freely write $\tau(u) = w$ instead of $(u, w) \in \tau$.

A string transducer is a tuple $M = \langle \Sigma, \Delta, Q, init, rul, fin \rangle$ where Σ and Δ are finite alphabet for input and output words, Q is a finite set of states, $init \subseteq Q$ is a set of initial states, $fin \subseteq Q \times \Delta^*$ the set of final states equipped

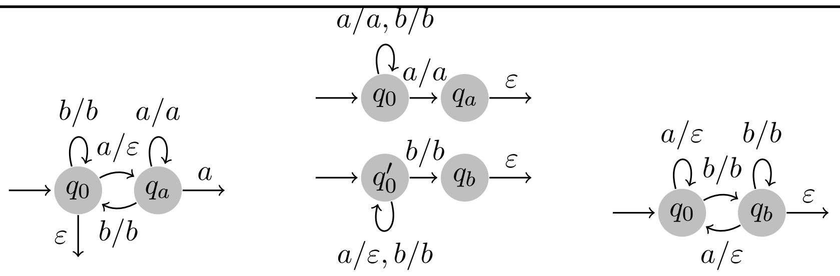


Figure 6.1: (a) A Dwt for τ_1. (b) A string transducer for τ_2. (c) A Dwt for τ_3.

with output words, and $rul \subseteq (Q \times \Sigma) \times (\Delta^* \times Q)$ is a finite set of transitions. We say that $q \xrightarrow{a/w} q'$ is a rule of M if $(q, a, w, q') \in rul$, and that $q \xrightarrow{w}$ is a final output if $(q, w) \in fin$. This arrow notion is also used in graphical representations of string transducers.

We denote by $[\![M]\!] \subseteq \Sigma^* \times \Delta^*$ the set of pairs (u, w) such that w is an output word that can be produced from input word u by M. More formally, a pair (u, w) belongs to $[\![M]\!]$ if there exists an index n, decompositions $u = a_1 \cdot \ldots \cdot a_n$ and $w = w_1 \cdot \ldots \cdot w_n \cdot w_f$, and a sequence of states $q_0 \cdot \ldots \cdot q_n$ such that $q_0 \in init$, $q_{i-1} \xrightarrow{a_i/w_i} q_i$ is a rule of M for all $1 \leq i \leq n$, and $q_n \xrightarrow{w_f}$ is a final output. A partial function is called rational if it is equal to $[\![M]\!]$ for some string transducer M, which is then called a functional transducer.

A string transducer is called deterministic or a Dwt (or subsequential) if it has at most one initial state and if rul and fin are partial functions. Clearly, every Dwt defines a rational function. Such functions are called subsequential, a notion going back to Schützenberger.

Example 139. *The total function τ_1 on words with alphabet $\{a, b\}$ that erases all a's immediately followed by b is subsequential. See Fig. 6.1 for a* Dwt *defining it. Notice that the final output is needed, for instance for transducing the word aa correctly to itself.*

The function τ_2 that deletes all a's in words whose last letter is b while performing the identity otherwise is rational, but not subsequential since the last letter cannot be predicted deterministically.

But if one restricts the domain of τ_2 to words ending by b, we obtain a partial function τ_3 which is subsequential, as illustrated in Fig. 6.1.

We denote by M_q the transducer equal to M except that q is the only initial state. A word $u \in \Sigma^*$ reaches a state q if there is a sequence of letters $a_1 \ldots a_n = u$ and of states $q_0 \ldots q_n$ such that $q_0 \in init$, $q_n = q$ and $q_{i-1} \xrightarrow{a_i/w_i} q_i$ is a rule of M for all $1 \leq i \leq n$ for some w_i. We call a Dwt M earliest if for all states q of M except the initial one, either the domain of $[\![M_q]\!]$ is the empty set or the least common prefix of all words in the range of $[\![M_q]\!]$ is the empty word.

Theorem 140 (Choffrut (1979) [Choffrut, 1979, Choffrut, 2003])**.** *Any subsequential function can be defined by some earliest* DWT*. The earliest* DWT *with a minimal number of states for a subsequential function is unique modulo state renaming.*

The DWTs in Fig. 6.1 (a) and (c) are both earliest and minimal. Note that a smaller single state DWT would be sufficient for defining τ_3 if the domain could be checked externally, which is not the case in this model.

Oncina and Varo [Oncina and Varo, 1996] used the Myhill-Nerode behind Theorem 140 as a theoretical ground for a learning algorithm for subsequential functions τ from a finite sample $S \subseteq \tau$ and a DFA A recognizing the domain of τ.

Theorem 141 (Oncina and Varo (1996))**.** *For any* DFA A *there exists a learning algorithm* OSTIA_A *that identifies subsequential functions whose domain is recognized by* A *from polynomial time and data.*

That is: for any DWT M defining a subsequential function τ whose domain is recognized by A there exists a finite sample $S \subseteq \tau$ called characteristic for τ, whose size is polynomial in the size of M, such that from any sample $S' \subseteq \tau$ that contains S, $\text{OSTIA}_A(S')$ computes a DWT defining τ in polynomial time in the size of S'.

6.3 Transducers with Look-Ahead

As stated before, rational functions are captured by deterministic transducers with look-ahead. The look-ahead can be performed by some DFA that annotates the letters of the input word by states from right to left in a preprocessing step. The string transducer then processes the annotated word from left to right. More formally, we can identify a DFA A with alphabet Σ and state set R with a string transducer that reads the word right to left, while always outputing the pair of the current letter and the current state: an automaton rule $r \xrightarrow{a} r'$ of A is considered as a transducer rule $r \xrightarrow{a/(a,r')} r'$. This way, the rational function $[\![A]\!]$ maps a word $u \in \Sigma^*$ to the identical word but annotated with look-ahead states $[\![A]\!](u) \in (\Sigma \times R)^*$. Furthermore, the DFA used as a lookahead must be complete, so that it defines a total function.

A deterministic string transducer with look-ahead (DWT_ℓ) is a pair $N = (A, M)$ such that A is a DFA with alphabet Σ and state set R called the look-ahead, and M is a DWT with signature $\Sigma \times R$ with state set Q. A DWT_ℓ $N = (A, M)$ defines the rational function $[\![N]\!] = [\![M]\!] \circ [\![A]\!]$: an input word $u \in \Sigma^*$ is first annotated with states of the look-ahead A from right to left, and then transformed by DWT M from left to right. The following theorem is known as the decomposition theorem of Elgot and Mezei [Elgot and Mezei, 1965].

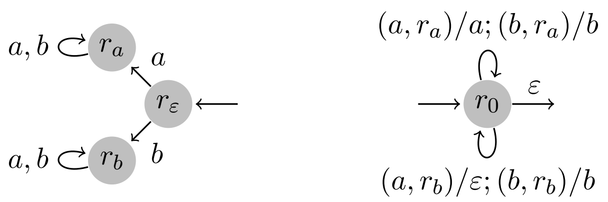


Figure 6.2: The look-ahead for τ_2 and a matching DWT.

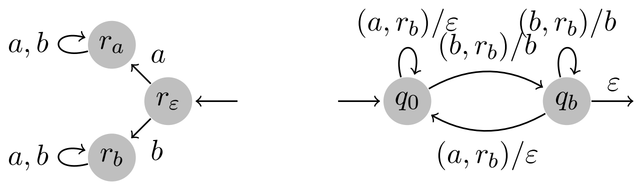


Figure 6.3: A look-ahead for τ_3, and a matching DWT, both compatible with their domains.

Theorem 142 (Elgot and Mezei (1965)). *A partial function τ is rational if and only if it is defined by some* DWT$_\ell$.

Given a string transducer M that defines a partial function, the idea is to use a look-ahead automaton to annotate positions by the set P of those states of M by which a final state can be reached at the end of the word. One can then define a DWT$_\ell$ N which simulates M except that it always selects an arbitrary transition leading to some state of P. Which of these transition is selected does not matter since M is functional

Example 143. *A* DWT$_\ell$ *for τ_2 is given in Fig. 6.2. Note that 3 look-ahead states are needed in order to distinguish suffixes ending with b or not.*

We next study the question of whether there exists a unique minimal lookahead automaton for any rational function. We obtain a positive result by reformulating a Myhill-Nerode style theorem for bi-machines from Reutenauer and Schütenberger [Reutenauer and Schützenberger, 1991].

A relation $\sim$ over $\Sigma^* \times \Sigma^*$ is called a left-congruence if $v_1 \sim v_2$ implies $u \cdot v_1 \sim u \cdot v_2$ for all input words v_1, v_2, u. Every look-ahead automaton A defines a left-congruence $\sim_A$ such that $v_1 \sim_A v_2$ if and only if v_1 and v_2 are evaluated to the same state by A (from the right to the left). Conversely, for any left-congruence $\sim$ with a finite number of equivalence classes, we can define a look-ahead automaton $A(\sim)$ such that $\sim$ is equal to $\sim_A$. The states of A are the equivalence classes $[u]_\sim$ of input words u, the unique initial state is the equivalence class of the empty word, and the transition rules have the form $[a \cdot u]_\sim \xleftarrow{a} [u]_\sim$ for all $u \in \Sigma^*$ and $a \in \Sigma$. Final states are irrelevant for look-ahead automata.

Domains of partial functions τ need to be treated carefully for look-ahead minimization. Let the left residual of its domain be $dom(\tau)v^{-1} = \{u \mid u \cdot v \in dom(\tau)\}$. The domain induces a left-congruence on suffixes that we call compatibility with the domain: v_1 and v_2 are compatible with the $dom(\tau)$ if $dom(\tau)v_1^{-1} = dom(\tau)v_2^{-1}$. A relation $\sim$ is said compatible with $dom(\tau)$ if it is a refinement of the compatibility relation, i.e., if $v_1 \sim v_2$ implies that v_1 and v_2 are compatible with $dom(\tau)$. Similarly, a look-ahead automaton A is compatible with a domain if $\sim_A$ is.

Let τ be a rational function. The difference between two output words is $diff(w \cdot w_1, w \cdot w_2) = (w_1, w_2)$ such that the common prefix of w_1 and w_2 is empty. The difference between two input words modulo τ is defined by $diff_\tau(v_1, v_2) = \{diff(\tau(u \cdot v_1), \tau(u \cdot v_2)) \mid u \cdot v_1,\ u \cdot v_2 \in dom(\tau)\}$. This allows to define a left-congruence $\sim_\tau$ that is compatible with $dom(\tau)$:

Definition 144. *$v_1 \sim_\tau v_2$ if and only if v_1 and v_2 are compatible with $dom(\tau)$ and $\#diff_\tau(v_1, v_2) < \infty$.*

Example 145. *The equivalence τ_1 has a single class since $diff_\tau(v_1, v_2)$ is finite for every $v_1, v_2 \in \Sigma^*$. Function τ_2 has two equivalence classes, since $v_1 \sim_{\tau_2} v_2$ if either both end with b or none. Indeed, $A(\sim_{\tau_2})$ is the look-ahead automaton in Fig. 6.2. Let $u_n = a^n \cdot b^n$. Then we have $\tau_2(u_n \cdot v_1) = u_n \cdot v_1$ while $\tau_2(u_n \cdot v_2) = b^n \cdot \tau_2(v_2)$. So $diff_{\tau_2}(v_1, v_2)$ contains the pairs $(a^n \cdot b^n \cdot v_1, b^n \cdot \tau_1(v_2))$ for all n, which as an infinite cardinality. Subsequential function τ_3 has 3 equivalence classes: a single state look-ahead automaton for τ_3 would not be compatible with the domain as for instance $dom(\tau_3)a^{-1} \neq dom(\tau_3)b^{-1}$. The* DWT$_\ell$ *with minimal look-ahead for τ_3 that is compatible with the domain has three states and is also the look-ahead given in Fig. 6.3. Note that neither the look-ahead nor the* DWT *are size minimal. Fig. 6.1 shows that there is no need for a look-ahead and Fig. 6.2 shows that for this look-ahead, τ_3 only needs a one-state* DWT. □

We say that a left congruence $\sim$ partitions $\sim_\tau$ if $\sim$ is a subset of $\sim_\tau$. For every partial function τ and an equivalence relation $\sim$ on Σ^*, we can define a unique partial function σ with minimal domain such that $\tau = \sigma \circ [\![A(\sim)]\!]$. This function σ, that we denote by $\tau\ \sim$, can be applied only to annotated words in the image of $[\![A(\sim)]\!]$; it ignores annotations and applies τ. The following result was originally stated for bimachines.

Theorem 146 (Reutenauer & Schützenberger [Reutenauer and Schützenberger, 1991])**.** *For any rational function τ the left-congruence $\sim_\tau$ has a finite number of equivalence classes. Furthermore, for any other left-congruence $\sim$ partitionning $\sim_\tau$ into finitely many classes, the function $\tau\ \sim$ is subsequential.*

As a result, any look-ahead for τ compatible with the domain of τ has the form $A(\sim)$ for some left-congruence $\sim$ that partitions $\sim_\tau$. Also, $\tau\ \sim$ being subsequential, Theorem 140 shows that it can be defined by a unique minimal DWT, that we denote by $M_\tau(\sim)$. The unique 'right-minimal' DWT$_\ell$ of τ then is the DWT$_\ell$ $N_\tau(\sim)$ equal to $\langle A(\sim), M_\tau(\sim)\rangle$.

6.4 Building the Look-Ahead Automaton

Our next objective is to find a suitable look-ahead automaton for the unknown target function τ, of which we only know the domain and a finite sample of input-output pairs. One might want to identify the minimal look-ahead automaton $A(\sim_\tau)$, but we cannot hope to decide whether $v_1 {\sim_\tau} v_2$ for any two words v_1 and v_2, since we would have to check whether $\mathit{diff}_\tau(v_1, v_2)$ is finite or infinite. This is difficult to archieve from a finite set of examples. We will work around this problem based on the following lemma which provides a bound on the cardinality of $\mathit{diff}_\tau(v_1, v_2)$.

Lemma 147. *Let $\tau \subseteq \Sigma^* \times \Delta^*$ be a rational function, $\sim$ a left congruence that partitions $\sim_\tau$ and m be the number of states of $M_\tau(\sim)$. If $v_1 \sim v_2$ then $\#\mathit{diff}_\tau(v_1, v_2) \leq m$.*

Proof. With $N = N_\tau(\sim)$, $v_1 \sim v_2$ implies $v_1 {\sim_\tau} v_2$, so that $dom(\tau)v_1^{-1} = dom(\tau)v_2^{-1}$. We denote by $[\![N]\!]^u(v)$ (resp. $[\![N]\!]_v(u)$) the output of v (resp. u) when reading $u \cdot v$. Then for any prefix $u \in dom(\tau)v_1^{-1}$, $\tau(u \cdot v_i) = [\![N]\!]_{v_i}(u) \cdot [\![N]\!]^u(v_i)$. By construction, $[\![N]\!]_{v_1}(u) = [\![N]\!]_{v_2}(u)$, so $\mathit{diff}(\tau(u \cdot v_1), \tau(u \cdot v_2))$ $= \mathit{diff}([\![N]\!]^u(v_1), [\![N]\!]^u(v_2))$. As $[\![N]\!]^u(v_i)$ only depends on the state reached by u in $A(\sim)$, the number of values of $([\![N]\!]^u(v_1), [\![N]\!]^u(v_2))$ for varying u is bounded by the number of states of $M_\tau(\sim)$, i.e. $\#\mathit{diff}_\tau(v_1, v_2) \leq m$. □ □

Given a natural number m we define the binary relation C_τ^m on input words such that $(v_1, v_2) \in C_\tau^m$ if $\#\mathit{diff}_\tau(v_1, v_2) \leq m$. In this case, we say that v_1 is m-close to v_2. As we will show in Section 6.6 for any m, we can characterize relation C_τ^m by finite samples of input-output pairs for τ.

Let m_τ be the number of states in $M_\tau(\sim_\tau)$. By Lemma 147 we know that $\sim_\tau = C_\tau^{m_\tau}$. So if we knew this bound m_τ and if we could construct a look-ahead automaton from $C_\tau^{m_\tau}$, then we were done. We first consider how to construct a look-ahead automaton from C_τ^m under the assumption that $m \geq m_\tau$.

Our algorithm LA given in Fig. 6.4 receives as inputs a binary relation $\mathcal{R}$ on input words and a natural number l, and returns as output a minimal deterministic finite automata, or raises an exception. Algorithm LA is motivated by the Myhill-Nerode theorem for deterministic finite automata, in that for l greater than the index of $\sim_\tau$ and $\mathcal{R} = C_\tau^{m_\tau} = \sim_\tau$ it constructs the minimal deterministic automaton $A(\sim_\tau)$. We will also apply it, however, in cases where $\mathcal{R}$ is even not an equivalence relation. In particular, relation

```
fun $\textsc{La}(\mathcal{R},l)$ // where $\mathcal{R} \subseteq \Sigma^* \times \Sigma^*$ , \ $l \in \mathbb{N}$
    let $Q = \textsc{Set}.new(\{\varepsilon\})$ , $Agenda = \textsc{Queue}.new([\varepsilon])$ in
    while \Agenda.isnonempty() do
      $v := Agenda.pop()$
      for $a \in \Sigma$ do
        if $\nexists v' \in Q$ such that $(a \cdot v, v') \in \mathcal{R}$
        then$Agenda.push(a \cdot v)$ , $Q.add(a \cdot v)$ else skip
        if $Q.card() > l$ then exception ``too many states''
           else skip
    let $rul = \{v \xrightarrow{a} v' \mid v, v' \in Q, (a \cdot v, v') \in \mathcal{R}\}$ in
    return $\langle \Sigma, Q, \{\varepsilon\}, \emptyset, rul \rangle$
```

Figure 6.4: Construction of look-ahead automata.

$\mathcal{R} = C_\tau^m$ may fail to be transitive for $m < m_\tau$. In this case we may have to force our algorithm to terminate. We do so by bounding the number of states that is to be generated by l.

Algorithm $\textsc{La}$ proceeds as follows. It fixes some total ordering on words, such that shorter words preceed on longer words. It then behaves as if $\mathcal{R}$ were a left congruences while searching for the least word in each equivalence class of $\mathcal{R}$. These least words will be the states of the output automaton that $\textsc{La}$ constructs. The algorithm raises an exception if the number of such states is greater then l. It adds the transitions $v \xrightarrow{a} v'$ for any two states v, v' that it discovered under the condition that $(a \cdot v, v') \in \mathcal{R}$ (if several v' fits, we pick the first in our order). We observe the following: if $\mathcal{R}$ is a left congruence of finite index smaller than l then $\textsc{La}(\mathcal{R}, l)$ terminates without exception and returns the minimal deterministic automata whose left-congruence is $\mathcal{R}$. In particular for $m \geq m_\tau$ and $\mathcal{R} = C_\tau^m$ (so that $\mathcal{R} = \sim_\tau$), the algorithm returns $A(\sim_\tau)$. However, if $m < m_\tau$, the only property that we can assume about relation C_τ^m is that it is contained in $\sim_\tau$. The following lemma shows a little surprisingly that successful result are always appropriate nevertheless.

Lemma 148. *Let τ be a rational function and $\mathcal{R}$ a relation contained in $\sim_\tau$. Either* $\textsc{La}(\mathcal{R}, l)$ *raises an exception or it returns a look-ahead valid for τ.*

If v_1 and v_2 are actually tested by the algorithm, then for v_1 and v_2 to be in the same state, we need $v_1 \mathcal{R} v_2$, and thus $v_1 \sim_\tau v_2$. Then, given that $\sim_\tau$ is a left-congruence, we can prove by recursion that if two words v_1 and v_2 reach the same state of $\textsc{La}(\mathcal{R}, l)$, then $v_1 \sim_\tau v_2$. Hence, $\mathcal{R}$ partitions $\sim_\tau$ so this $\textsc{La}(\mathcal{R}, l)$ is a valid look-ahead for τ by Theorem 146.

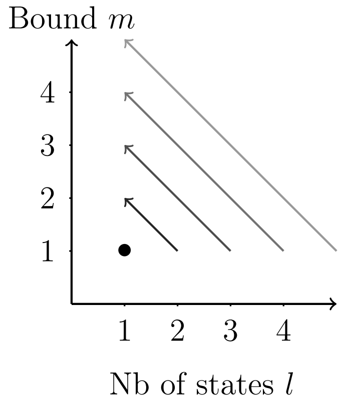
```
// Let $A_{dom}$ be a DFA
fun $\text{Learn}_{A_{dom}}(S)$
    let $(m,l) := (1,1)$
    repeat
      try let $A = \text{La}(C^m_{S,A_{dom}}, l)$ in
        let $S' = \{([\![A]\!](u), v) \mid (u,v) \in S)\}$ in
        let $A'$ be a DFA that represents words of $L(A_{dom})$ annotated by $A$ in
        return $\langle A, \text{Ostia}_{A'}(S')\rangle$
        exit
      catch ''too many states'' then
      $(m,l) :=$ successor of $(m,l)$ in diagonal order
```

Figure 6.5: Learning algorithm for rational functions of domain $L(A_{dom})$.

6.5 The Learning Algorithm

We next present an algorithm for learning a rational function τ from a domain automata A_{dom} with $L(A_{dom}) = dom(\tau)$ and a finite sample $S \subseteq \tau$ of input-output pairs. Furthermore, our learning algorithm assumes that there exists an oracle $C^m_{S,A}$ that can decide whether a pair of input words belongs to C^m_τ. Given such an oracle, the learning algorithm can simulate calls of algorithm $\text{La}(C^m_\tau, l)$. How such an oracle can be obtained for sufficiently rich samples S is shown in the next section.

Two unknowns remain to be fixed: a bound m for which La eventually finds a valid look-ahead and the number l of states of this valid look-ahead. The idea of learning algorithm Learn_A in Fig. 6.5 is that to try out all pairs (m,l) in diagonally increasing order $(1,1) < (1,2) < (2,1) < (1,3) < \ldots$. For any such pair (m,l) it then calls $\text{La}(C^m_{S,A}, l)$, until this algorithm succeeds to return an automaton. By Lemma 148, any such automaton is a valid look-ahead for τ. By Proposition 147, this procedure must be successful no later than for (m_τ, l_τ). Finally, the algorithm decorates the examples of S by applying the newly obtained look-ahead automaton, and learns the corresponding subsequential transducer by using the Ostia algorithm.



It should be noticed that the target of this algorithm is *not* the Dwt_ℓ for τ with minimal look-ahead $A(\sim_\tau)$. The look-ahead obtained is simply the first automaton obtained in the diagonal order such that $\text{La}(C^m_{S,A}, l)$ terminates successfully. We call the Dwt_ℓ obtained in this way the 'diagonal' Dwt_ℓ of τ. Note that the diagonal Dwt_ℓ of τ may be smaller that the corresponding

```
fun $C^m_{S,A}(v_1, v_2)$
    if $L(A)v_1^{-1} \neq L(A)v_2^{-1}$ then return false
    else if $\#\{\mathit{diff}(w_1, w_2) \mid (u \cdot v_1, w_1), (u \cdot v_2, w_2) \in S\} \leq m$
    then return true else return false
```

Figure 6.6: Implemention of the oracle.

right-minimal DWT_ℓ with minimal look-ahead. In any case, it may not be much bigger as stated by the following lemma.

Lemma 149. *Let τ be a partial rational function with right-minimal* DWT_ℓ *$\langle A(\sim_\tau), M(\sim_\tau)\rangle$, let m be the number of states of $M(\sim_\tau)$, and $\sim$ be a finite left-congruence that partitions $\sim_\tau$ of index n. The number of states of the look-ahead of $\langle A(\sim), M(\sim)\rangle$ has then at most mn states and is of global size $O(mn^2)$.*

Indeed, to obtain the DWT $M(\sim)$, one can pick $M(\sim_\tau)$ and change its transition to take into account states of $A(\sim)$ instead of those of $A(\sim_\tau)$. This transducer has m states and at worse mn transitions. However, it does not have the right domain (words annotated by states of $M(\sim)$): this requires a product with the DFA of the correct domain, which has m states. The actual DWT $M(\sim)$ being minimal, it has at most this size.

6.6 Characteristic Samples

It remains to show that there exists an oracle $C^m_{S,A}$ that decides membership to C^m_τ for all suffuciently rich finite samples $S \subseteq \tau$, and that the size of such samples is polynomial in the size of the target diagonal transducer with look-ahead. We use the function defined in Fig. 6.6 which when applied to a pair of words (v_1, v_2) verifies that they have equal residuals for the domain, and computes their difference on S instead of τ. In order to see that the former can be done in polynomial time, we only need to check that there are deterministic automata recognizing $L(A)v_1^{-1}$ and $L(A)v_2^{-1}$ of polynomial size.

The next question is what examples a sample S needs to contain so that this test becomes truly equivalent to m-closeness. In order to be usable in LA, note that $C^m_{S,A}(v_1, v_2)$ has to behave like $C^m_\tau(v_1, v_2)$ only on pairs of suffixes considered there. We define $s_{m,l}(\tau)$ as the words creating new states in $\text{LA}(C^m_\tau(v_1, v_2), l)$ (there is at most l of them). As the algorithm LA also observes successors of $s_{m,l}$, we need to define the set $k_{m,l}(\tau) = s_{m,l}(\tau) \cup \{a \cdot v \mid v \in s_{m,l}(\tau), a \in \Sigma\}$. We call a sample S ℓ-characteristic for τ with respect to m and l if every element of $k_{m,l}$ appears as the suffix of an input word in S and if S allows the correct evaluation of C^m_τ on those elements, i.e.:

- for every $v \in s_{m,l}(\tau)$, $\exists u \in \Sigma^*$, $w \in \Delta^*$ such that $(u \cdot v, w) \in S$,
- for $v_1 \in s_{m,l}$, $v_2 \in k_{m,l}$ with $(v_1, v_2) \notin C_\tau^m$ and $dom(\tau)v_1^{-1} = dom(\tau)v_2^{-1}$, $\#\{diff(w_1, w_2) \mid (u \cdot v_1, w_1), (u \cdot v_2, w_2) \in S\} > m$.

Lemma 150. *For a partial rational function τ, a* DFA *A recognizing $dom(\tau)$, and two positive integers m and l, let $v_1 \in s_{m,l}(\tau)$, $v_2 \in k_{m,l}(\tau)$, if S is a ℓ-characteristic sample for τ with respect to m and l, then the test $C_{S,A}^m(v_1, v_2)$ returns true if and only if $(v_1, v_2) \in C_\tau^m$.*

One thing that has to be checked is that there exists an ℓ-characteristic samples of reasonable size for any m, l. This is obvious for the cardinality. In order to show that the length of words can also be guaranteed to be short, one can use the following method: for any non-equivalent suffixes v_1 and v_2 of different domain, one pick any set of words that allow to obtain enough element in $diff_\tau(v_1, v_2)$, and reduce them to a reasonable length (of size $\mathcal{O}(|N|^2)$) where N is any transducer recognizing τ) using pumping arguments.

Lemma 151. *For a partial rational function τ, a* DFA *A recognizing $dom(\tau)$, two integers m and l, and a sample S ℓ-characteristic for τ with respect to m and l:* $\text{LA}(C_{S,A}^m, l) = \text{LA}(C_\tau^m, l)$.

In particular, if $\text{LA}(C_{S,A}^m, l)$ raises an exception if and only if $\text{LA}(C_\tau^m, l)$ does. Note that we need a sample that is (globally) ℓ-characteristic, for all pairs $\langle m, l\rangle$ encountered during the run, i.e. all the $\langle m, l\rangle$ smaller than the values for the diagonal DWT_ℓ. Once the look-ahead is learned, we can apply the OSTIA algorithm, which requires a sample labelled by the look-ahead, and not on $\Sigma^* \times \Delta^*$. We deal with this by labelling all the input words in S when the look-ahead $A(\sim)$ is found. For S to be enough to learn the subsequential transducer $M_\tau(\sim)$, its labelling must contain a characteristic sample for the OSTIA algorithm as defined in [Oncina and Varo, 1996]. In other words, S is called DWT-characteristic for τ and $\sim$ if it contains a characteristic sample for $M_\tau(\sim)$ in OSTIA, minus the labelling by $\sim$.

Finally, for the algorithm LEARN_A to produce the diagonal DWT_ℓ, the input sample needs to be ℓ-characteristic. Also, it has to be DWT-characteristic for τ and the look-ahead $\sim$ it found. A sample S is then said to be characteristic for a rational function τ if it fulfils all those conditions. This gives the following result:

Theorem 152. *For any* DFA *A the learning algorithm* LEARN_A *identifies rational functions with domain $L(A)$ represented by their diagonal* DWT_ℓ *from polynomial time and data.*

That is: for any DWT_ℓ N in diagonal form defining a rational function τ whose domain $L(A)$, there exists a finite sample $S \subseteq \tau$ called characteristic for τ whose size is polynomial in the size of N, such that from any sample

$S' \subseteq \tau$ that contains S, $\textsc{Learn}_A(S')$ computes a $\textsc{Dwt}_\ell$ in diagonal-minimal normal form defining τ in polynomial time in the size of S'.

Conclusion and Future Work. Our learning algorithm for $\textsc{Dwt}_\ell$ s answers the long standing open learning question for rational functions, for the case where diagonal-minimal $\textsc{Dwt}_\ell$ normal forms are used for their representation. Whether other representations lead to negative results is left open. More importantly, we would like to extend our result to deterministic top-down tree transducers with look-ahead, which have the same expressiveness than functional top-down tree transducers [Engelfriet, 1977].

Chapter 7

Normal Forms of Linear Tree-to-Word Transducers

Abstract. We study a subclass of tree-to-word transducers: linear tree-to-word transducers, that cannot use several copies of the input. We aim to study the equivalence problem on this class, by using minimization and normalization techniques. We identify a Myhill-Nerode characterization. It provides a minimal normal form on our class, computable in Exptime. This chapter extends an already existing result on tree-to-word transducers without copy or reordering (sequential tree-to-word transducers), by accounting for all the possible reorderings in the output.

Contents

7.1 Introduction **138**
7.2 Linear Tree-to-Word Transducers **139**
7.3 Earliest Linear Transducers **141**
7.4 Reordering in Earliest Transducers **143**
7.4.1 Reordering Erasing States 144
7.4.2 Reordering Producing States 145
7.5 Myhill-Nerode Theorem and Normal Form **148**
7.6 Conclusion and Future Work **149**

7.1 Introduction

The deterministic top-down tree transducers studied in Chapter 4 and 5 can be seen as top-down rewriting systems for trees. This means they lack some important expressive power that one could desire from an object that describe tree transformations. One such power is concatenation in the output, that plays an important role in the way XSLT produces its outputs. Indeed, XSLT can describe not only transformations from trees to trees, but also from trees to hedges. Two such tree-to-hedge functions can be “combined” into a third XSLT-described transformation that produces the hedge concatenation of their productions.

To model this concatenation power with tree transducers is a highly challenging problem. A reasonable approach would be to study the consequences of concatenation in the output for simpler classes first. The class of tree-to-word transducers is of particular relevence, as it can be seen as a tool to model the very particular case where each tree in the hedge produced is a single symbol. They are notably stable under concatenation. The study of tree-to-word transducers is also relevant as it can be seen as a restricted case of Macro-tree transducers [Engelfriet and Vogler, 1985], a tree transducer class whose extensive expressiveness make for a desirable, if difficult, class to study. In fact, if a Macro-tree transducer were to produce on an unary signature, its power would possess the concatenation power described in tree-to-word transducers.

This new concatention power, however, appears to be difficult to combine with the classical techniques of language theory, which leads to few results on this class as a whole. Equivalence for all tree-to-word transducers has recently been proven to be decidable [Seidl et al., 2015] with a co-randomized polynomial algorithm for the linear case. Note that this result uses neither classic logic methods, nor the classic transducer methods, and does not provide a characterization or Myhill-Nerode theorem.

One particular fragment of the class of tree-to-word transducers has been studied, and provided with several results that successfully extends classic transducer methods to transducers with computation in the output: sequential tree-to-word transducers (or STWs), that prevents copying in the output and forces subtrees to produce following the order of the input. Equivalence is PTIME for sequential tree-to-word transducers [Laurence et al., 2011]. Furthermore, using a *Myhill-Nerode* characterization, a normal form computable in EXPTIME is shown to exist. This normal form was later proven to be learnable in PTIME [Laurence et al., 2014].

In this chapter, we aim to study the linear tree-to-word transducers (or LTWs), a restriction of deterministic tree-to-word transducers that forbids copying in the output, but allows the image of subtrees to be flipped in any order. This is a more general class than sequential tree-to-word transducers, but still less descriptive than general tree-to-word transductions. In this class, we show the existence of a normal form, computable in EXPTIME.

Note that even if equivalence is already known to be decidable in a reasonable complexity, finding a normal form is of general interest in and of itself. For example, in [Oncina et al., 1993, Lemay et al., 2010, Laurence et al., 2014], normal forms on transducers defined using a Myhill-Nerode theorem are used to obtain a learning algorithm.

To define a normal form on LTWS, we start by the methods used for sequential tree-to-words transducers (STWS) in [Laurence et al., 2011]. We consider the notion of *earliest* STWS, which normalizes the output production. We can extend this notion to LTWS and study only earliest LTWS without any loss of expressivity.

In [Laurence et al., 2011], this is enough to obtain a Myhill-Nerode characterization. However, by adding the possibility to flip subtree images to LTWS, we created another way for equivalent transducers to differ. The challenge presented by the extension of the methods of [Laurence et al., 2011] becomes to resolve this new degree of freedom, in order to obtain a good normal form with a Myhill-Nerode characterization.

Outline. We will present our class of linear tree-to-word transducers in Section 7.2. Then in Section 7.3 we will extend the notion of *earliest* production in [Laurence et al., 2011] to the linear case, and find out that we can also extend the algorithm that takes a transducer and compute and equivalent earliest one. However, this is no longer sufficient, as transducers can now also differ in the order they produce their subtrees' output in. Section 7.4 will detail exactly how two earliest transducers can still differ, by categorizing all possible flips. Finally, Section 7.5 will compile these results into a Myhill-Nerode theorem. This will allow us to establish a normal form, computable in EXPTIME. We will conclude by a brief recap of the result, and propose several possible next steps for this line of research.

7.2 Linear Tree-to-Word Transducers

We consider a finite tree signature Σ as well as a finite word alphabet Δ. We define linear tree-to-word transducer, that define a function from $\mathcal{T}$ to Δ^*.

Definition 153. *A linear tree-to-word transducer (*LTW*) is a tuple* $M = \{\Sigma, \Delta, Q, ax, rul\}$ *where*

- Σ *is a tree alphabet,*
- Δ *is a finite word alphabet of output symbols,*
- Q *is a finite set of states,*
- ax *is a axiom of form* $u_0 q u_1$*, where* u_0, $u_1 \in \Delta^*$ *and* $q \in Q$*,*

- *rul is a set of rules of the form*

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$$

where $q, q_1, \dots, q_n \in Q$, $f \in \Sigma$ *of rank* n *and* $u_0 \dots u_n \in \Delta^*$; σ *is a permutation on* $\{1, \dots, n\}$. *There is at most one rule per pair* q, f.

We define recursively the function $[\![M]\!]_q$ *of a state* q. $[\![M]\!]_q(f(t_1 ... t_n))$ *is*

- $u_0 [\![M]\!]_{q_1}(t_{\sigma(1)}) u_1 \dots [\![M]\!]_{q_n}(t_{\sigma(n)}) u_n$,
  if $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n \in \delta$
- *undefined, if there is no rule for* q, f *in* δ.

The function $[\![M]\!]$ *of a transducer* M *with axiom* $u_0 q u_1$ *is defined as* $[\![M]\!](s) = u_0 [\![M]\!]_q(s) u_1$.

Note that to get the definition of STWs as made in [Laurence et al., 2011], we just have to impose that in every rule, σ is the identity.

Example 154. *Consider the function* $[\![M]\!] : t \mapsto 0^{|t|}$, *that counts the number of nodes in* t *and writes a* 0 *in the output for each of them. Our* LTW *has only one state* q, *and its axiom is* $ax = q$

$$q(f(x_1, x_2)) \to 0 \cdot q(x_1) \cdot q(x_2)$$
$$q(a) \to 0, \quad q(b) \to 0$$

The image of $f(a, b)$ *is* $[\![M]\!](f(a, b)) = [\![M]\!]_q(f(a, b))$, *using the axiom. Then we use the first rule to get* $0 \cdot [\![M]\!]_q(a) \cdot [\![M]\!]_q(b)$, *and finally,* $0 \cdot 0 \cdot 0$

We denote with $dom([\![M]\!])$ the domain of a transducer M, i.e. all trees such that $[\![M]\!](t)$ is defined. Similarly, $dom([\![M]\!]_q)$ is the domain of state q.

We define accessibility between states as the transitive closure of appearance in a rule. This means q is accessible from itself, and if there is a rule $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$, and q accessible from q', then all states q_i, $1 \leq i \leq n$, are accessible from q'.

We note L_q the set of all productions of q: $L_q = \{[\![M]\!]_q(t) | t \in dom([\![M]\!]_q)\}$. We call a state *periodic* of period $w \in \Delta^*$ if $L_q \subseteq w^*$.

We start the normalization process with a natural notion of trimmed LTWs.

Definition 155. *A* LTW *is* trimmed *if its axiom is* $u_0 q_0 v_0$, *and every state* q *is accessible from* q_0 *and of non-empty domain.*

Note that all LTWs can be made trimmed by deleting all their useless states.

Lemma 156. *For* M *a* LTW, *one can compute an equivalent trimmed* LTW *in linear time.*

7.3 Earliest Linear Transducers

It is possible for different LTWs to encode the same transformation. To reach a normal form , we start by requiring our LTWs to produce their output "as soon as possible". This method is common for transducers [Choffrut, 2003, Engelfriet et al., 2009], and has been adapted to sequential tree-to-word transducers in [Laurence et al., 2011]. In this case, the way an output word is produced by a tree-to-word can be "early" in two fashions: it can be produced sooner in the input rather than later, or it can output letters on the left of a rule rather than on the right. We take the natural extension of this definition for LTWs and find we can reuse the results and algorithms of [Laurence et al., 2011].

Example 157. *Consider our previous example (Ex. 154). The function* $[\![M]\!] : t \mapsto 0^{|t|}$*, Our transducer has only one state* q*, and its axiom is* $ax = q$

$$q(f(x_1, x_2)) \to 0 \cdot q(x_1) \cdot q(x_2)$$
$$q(a) \to 0, \quad q(b) \to 0$$

Since all productions of q *start with a* 0*, this* LTW *does not produce its first* 0 *in an earliest manner. To change this, we form a new state* q' *that produces one* 0 *less than* q*. By removing the* 0 *at the beginning of each rule of* q*, and replacing each call* $q(x_i)$ *by* $0q'(x_i)$*, we get a new equivalent* LTW M' *of axiom* $ax' = 0 \cdot q'$

$$q'(f(x_1, x_2)) \to 0 \cdot q'(x_1) \cdot 0 \cdot q'(x_2)$$
$$q'(a) \to \varepsilon \quad q'(b) \to \varepsilon$$

Example 158. *Consider our previous example (Ex. 157). We could replace the first rule by* $q'(f(x_1, x_2)) \to 0 \cdot 0 \cdot q'(x_1) \cdot q'(x_2)$*. This new* LTW *would produce "more to the left", but still be equivalent to the first* M*.*

In order to eliminate these differences in output strategies, we want transducers to produce the output as up in the input tree as possible, and then as to the left as possible. We formalize these notions in the definition of *earliest* LTWs.

To simplify notations, we note $lcp(q)$ (or $lcs(q)$) for $lcp(L_q)$ (or $lcs(L_q)$). By extension, for $u \in \Delta^*$, we note $lcp(qu)$ (or $lcs(qu)$) for $lcp(L_q.u)$ (or $lcs(L_q.u)$).

Definition 159. *A* LTW M *is* earliest *if it is trimmed, and:*

- *For every state* q*,* $lcp(q) = lcs(q) = \varepsilon$
- *For each rule* $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n \in rul$*, for every* i *from* 1 *to* n*,* $lcp(q_i u_i) = \varepsilon$

This definition is a generalization of the one found in [Laurence et al., 2011] from STWs to all LTWs. The first item ensures an earliest LTW outputs as soon as possible, the second that it produces as to the left as possible. Note that this means that $u_0 q_1(x_{\sigma(1)}) ... q_i(x_{\sigma(i)}) u_i$ produces as much of $\llbracket M \rrbracket_q(f(s_1...s_n))$ by just knowing $s_{\sigma(1)}, ..., s_{\sigma(i)}$, i.e. the *lcp* of all $\llbracket M \rrbracket_q(f(s_1...s_n))$ for some fixed $s_{\sigma(1)}, ..., s_{\sigma(i)}$.

Lemma 160. *For M an earliest* LTW*, $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n \in$ rul,*
for i such that $i \leq n$, $t_{\sigma(1)}, ..., t_{\sigma(i)}$ respectively in dom($\llbracket M \rrbracket_{q_1}$), ..., dom($\llbracket M \rrbracket_{q_i}$), then $u_0 \llbracket M \rrbracket_{q_1}(t_{\sigma(1)}) ... \llbracket M \rrbracket_{q_i}(t_{\sigma(i)}) u_i$ is the lcp of the set:

$$\left\{ \llbracket M \rrbracket_q(f(s_1, ...s_n)) | s_{\sigma(1)} = t_{\sigma(1)}, ..., s_{\sigma(i)} = t_{\sigma(i)} \right\}$$

.

In intuition, this comes from the fact that in an earliest, on the right of $u_0 \llbracket M \rrbracket_{q_1}(t_{\sigma(1)}) ... \llbracket M \rrbracket_{q_i}(t_{\sigma(i)}) u_i$, one cannot guess the first letter of $\llbracket M \rrbracket_{q_{i+1}}(t_{\sigma(i+1)}) ... \llbracket M \rrbracket_{q_n}(t_{\sigma(n)}) u_n$.

Some important properties extend from [Laurence et al., 2011] to earliest LTWs, most notably the fact that all LTWs can be made earliest.

Lemma 161. *For M a* LTW*, one can compute an equivalent earliest* LTW *in exponential time.*

This result is a direct generalization of the construction in Section 3 of [Laurence et al., 2011]. We build the equivalent earliest LTW M' with two kinds of steps:

- If *lcp*$(qu) = v$, where v is a prefix of u, we can slide v through state q by creating a new state $[v^{-1}qv]$ such that for all t, $\llbracket M' \rrbracket_{[v^{-1}qv]}(t) = v^{-1} \llbracket M \rrbracket_q(t) v$. Every occurrence of $q(x_i)v$ in a rule of M is replaced by $v \left[v^{-1}qv\right](x_i)$.

- If *lcp*$(q) = v$, we can produce v outside of q by creating a new state $[v^{-1}q]$ such that for all t, $\llbracket M' \rrbracket_{[v^{-1}q]}(t) = v^{-1} \llbracket M \rrbracket_q(t)$. Every occurrence of $q(x_i)$ in a rule of M is replaced by $v \left[v^{-1}q\right](x_i)$.
  Symmetrically, if *lcs*$(q) = v$, we create a state $[qv^{-1}]$, and every occurrence of $q(x_i)$ in a rule of M is replaced by $\left[qv^{-1}\right](x_i)v$.

Note that the exponential bound is, in fact, an exact bound, as some LTWs gain an exponential number of states through this process.

In [Laurence et al., 2011], earliest STWs are actually enough to make a normal form using a Myhill-Nerode theorem: by minimizing earliest STWs (merging states with the same $\llbracket M \rrbracket_q$), we end up with a normal form with a minimal number of states. However, in the wider case of LTWs, there are still ways for two states to be equivalent and yet not syntactically equal. This impedes the process of minimization. As we will see in the next part, it remains to study how the images of subtrees can be reordered in earliest LTWs while preserving equivalence.

7.4 Reordering in Earliest Transducers

Syntactically different earliest LTWs may still be equivalent. Indeed, unlike sequential tree transducers [Laurence et al., 2011], which impose the output to follow the order of the input, LTWs permit to flip the order.

The main point of this chapter is the observation that it is sufficient to normalize the flips in the output production of earliest LTWs, in order to find a unique normal form for equivalent LTWs. To this end, we will prove that order differences are only possible in very specific cases. We start illustrating such flips in some examples, and then discuss the necessary and sufficient condition that dictates when a flip is possible.

Example 162. *We reconsider Example 158 . This earliest transducer "counts" the number of nodes in the input tree has only one state q'. It has the axiom $ax' = 0 \cdot q'$ and the following rules:*

$$q'(f(x_1, x_2)) \to 0 \cdot 0 \cdot q'(x_1) \cdot q'(x_2), \qquad q'(a) \to \varepsilon, \qquad q'(b) \to \varepsilon.$$

We can now flip the order of the terms $q'(x_2)$ and $q'(x_1)$ in the first rule, and replace it by:

$$q'(f(x_1, x_2)) \to 0 \cdot 0 \cdot q'(x_2) \cdot q'(x_1).$$

This does not change $[\![M']\!]$, since just the order is changed in which the nodes of the first and second subtree of the input are counted.

Of course, it is not always possible to flip two occurrences of terms $q_1(x_{\sigma(1)})$ and $q_2(x_{\sigma(2)})$ in LTW rules.

Example 163. *Consider an earliest transducer that outputs the frontier of the input tree while replacing a by 0 and b by 1. This transducer has a single state q, the axiom $ax = q$, and the following rules:*

$$q(f(x_1, x_2)) \to q(x_1) \cdot q(x_2), \qquad q(a) \to 0, \qquad q(b) \to 1.$$

Clearly, replacing the first rule by a flipped variant $q(f(x_1, x_2)) \to q(x_2) \cdot q(x_1)$ would not preserve transducer equivalence since $f(a, b)$ would be transformed to 10 instead of 01. More generally, no LTW with rule $q(f(x_1, x_2)) \to u_0 \cdot q_1(x_2) \cdot u_1 \cdot q_2(x_1) \cdot u_2$ produces the correct output.

Our goal is to understand the conditions when variable flips are possible.

Definition 164. *For M, M' two* Ltws*, $q \in Q$, $q' \in Q'$,*

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n \in rul$$

$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)}) u'_n \in rul'$$

are said to be twin rules *if q and q' are equivalent.*

7.4.1 Reordering Erasing States

We start the study of possible reordering with the obvious case of states that only produce ε: they can take every position in every rule without changing the semantics of the states. The first step towards normalization would then be to fix the positions of erasing states in the rules, to prevent differences in equivalent earliest Ltws: we put all erasing states at the end of any rule they appear in, in ascending subtree order.

Definition 165. *For M a* Ltw*, a state q is* erasing *if for all $t \in dom(\llbracket M \rrbracket_q)$, $\llbracket M \rrbracket_q(t) = \varepsilon$*

We show that if two states are equivalent, they call erasing states on the same subtrees. We start by this length consideration:

Lemma 166. *For two twin rules of earliest* Ltws

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$$

$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)}) u'_n$$

For i, j such that $\sigma(i) = \sigma'(j)$, and $t_{\sigma(i)} \in dom(\llbracket M \rrbracket_{q_i})$ then $|\llbracket M \rrbracket_{q_i}(t_{\sigma(i)})| = |\llbracket M' \rrbracket_{q'_j}(t_{\sigma(i)})|$

Proof. The equivalence of q and q' gives for all $t_1, ..., t_n$:

$$u_0 \llbracket M \rrbracket_{q_1}(t_{\sigma(1)}) ... \llbracket M \rrbracket_{q_n}(t_{\sigma(n)}) u_n = u_0 \llbracket M' \rrbracket_{q'_1}(t_{\sigma'(1)}) ... \llbracket M' \rrbracket_{q'_n}(t_{\sigma'(n)}) u'_n$$

By fixing every t_k except $t_{\sigma(i)}$ we get that for some u, v, u', v', $u \llbracket M \rrbracket_{q_i}(t_{\sigma(i)}) v = u' \llbracket M' \rrbracket_{q'_j}(t_{\sigma(i)}) v'$. If $|\llbracket M \rrbracket_{q_i}(t_{\sigma(i)})| > |\llbracket M' \rrbracket_{q'_j}(t_{\sigma(i)})|$ then $|u| < |u'|$, or $|v| < |v'|$. If $|u| < |u'|$, then $u' = uw$. For all $t_{\sigma(i)}$, $\llbracket M \rrbracket_{q_i}(t_{\sigma(i)}) \neq \varepsilon$ (it is longer than $\llbracket M' \rrbracket_{q'_j}(t_{\sigma(i)})$), and its first letter is always the first letter of w. This means $lcp(q_i) \neq \varepsilon$, which is impossible in an earliest Ltw. $|v| < |v'|$ leads to $lcs(q_i) \neq \varepsilon$, another contradiction. By symmetry, $|\llbracket M' \rrbracket_{q'_j}(t_{\sigma(i)})| > |\llbracket M \rrbracket_{q_i}(t_{\sigma(i)})|$ also leads to contradiction. Therefore, both are of same size. □

Lemma 167. *For two twin rules of earliest* Ltws

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$$

$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)}) u'_n$$

For i, j such that $\sigma(i) = \sigma'(j)$, If q_i is erasing, then q'_j is erasing.

To normalize the order of erasing states in twin rules, we note that since an erasing state produces no output letter, its position in a rule is not important to the semantics or the earliest property. We can thus push them to the right.

Lemma 168. *For M an earliest* Ltw*, $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$ a rule in M, and q_i an erasing state. Then replacing this rule by*

$$q, f \to u_0 q_1(x_{\sigma(1)}) ... u_{i-1} u_i ... q_n(x_{\sigma(n)}) u_n q_i(x_{\sigma(i)})$$

does not change $[\![M]\!]_q$, and M remains earliest.

Note that the earliest property also imposes that if q_i is erasing, $u_i = \varepsilon$.

Given this lemma, we can define a first normalization step where all erasing states appear at the end of the rules in ascending subtree order.

Definition 169. *An earliest* Ltw *M is* erase-ordered *if for every rule $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n \in rul$, if q_i is erasing, then for all $j > i$, q_j is erasing, and $\sigma(i) < \sigma(j)$.*

Lemma 170. *For M an earliest* Ltw*, one can make M erase-ordered in polynomial time without changing the semantic of its states.*

We can detect if a state q is erasing by checking that no accessible rule produces a letter. From there, Lemma 168 ensures that making a Ltw erase-ordered is just a matter of pushing all erasing states at the end of the rules and them sorting them in ascending subtree order.

7.4.2 Reordering Producing States

As we saw in Example 163, some flips between states are not possible. We will now study what makes reordering non-erasing states possible. As we will see, only few differences are possible between twin rules in erase-ordered earliest Ltws. Two states transforming the same subtree are equivalent, and the only order differences are caused by flipping states whose productions commute in Δ^*.

To prove this, we begin by establishing a few preliminary results. We first show that to the left of σ and σ''s first difference, both rules are identical.

Lemma 171. *For two twin rules of erase-ordered earliest* Ltw*s M, M'*

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$$

$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)}) u'_n$$

For i such that if $k \leq i$ then $\sigma(k) = \sigma'(k)$, $[\![M]\!]_{q_i} = [\![M']\!]_{q'_i}$, and $u_i = u'_{i'}$

Proof. This results from Lemma 160: if σ and σ' coincide before i, then for all $t_{\sigma(1)}, ..., t_{\sigma(i)}$, $u_0[\![M]\!]_{q_1}(t_{\sigma(1)})...u_i$ and $u'_0[\![M']\!]_{q_1}(t_{\sigma'(1)})...u'_i$ are both equal to the *lcp* of $\{[\![M]\!]_q(f(s_1,...,s_n))|s_{\sigma(1)} = t_{\sigma(1)}, ..., s_{\sigma(n)} = t_{\sigma(n)}\}$. This means that:

$$u_0[\![M]\!]_{q_1}(t_{\sigma(1)})...[\![M]\!]_{q_i}(t_{\sigma(i)})u_i = u_0[\![M']\!]_{q'_1}(t_{\sigma'(1)})...[\![M']\!]_{q'_i}(t_{\sigma'(i)})u'_i$$

Since this is also true for $i-1$, we can remove everything but the last part for each side of this equation, to obtain that for all $t_{\sigma(i)}$, $[\![M]\!]_{q_i}(t_{\sigma(i)})u_i = [\![M']\!]_{q'_i}(t_{\sigma(i)})u'_i$. Lemma 166 gives us $|[\![M]\!]_{q_i}(t_{\sigma(i)})| = |[\![M']\!]_{q'_i}(t_{\sigma'(i)})|$, and $u_i = u'_i$. This means that q_i and q'_i are equivalent, and $u_i = u'_i$. □

It still remains to show what happens when σ and σ' stop coinciding. We study the leftmost order difference between two twin rules in erasing-ordered earliest LTWs, that is to say the smallest i such that $\sigma(i) \neq \sigma'(i)$. Note that Lemma 167 ensures that such a difference occurs before the end of the rule where the erasing states are sorted.

Lemma 172. *For two twin rules of erase-ordered earliest* LTW*s M, M'*

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)})u_n$$

$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)})u'_n$$

For i such that $\sigma(i) \neq \sigma'(i)$ and for any $k < i$, $\sigma(k) = \sigma'(k)$, for j such that $\sigma'(i) = \sigma(j)$, we have:

(A) For all k from i to $j-1$, $u_k = \varepsilon$ and there exists a tree $t^{\varepsilon}_{\sigma(k)}$ such that $[\![M]\!]_{q_k}(t^{\varepsilon}_{\sigma(k)}) = \varepsilon$

(B) For all k from i to j, for k' such that $\sigma(k) = \sigma'(k')$, q_k is equivalent to $q'_{k'}$

(C) All $q_i, ..., q_j$ are periodic of same period.

As a proof intuition, we first prove point (A), then use it to show point (B), then from (A) and (B) we finally show point (C).

For point (A), we use the equivalence of q and q'. For all $t_1, ..., t_n$,

$$u_0[\![M]\!]_{q_1}(t_{\sigma(1)})...[\![M]\!]_{q_n}(t_{\sigma(n)})u_n = u_0[\![M']\!]_{q'_1}(t_{\sigma'(1)})...[\![M']\!]_{q'_n}(t_{\sigma'(n)})u'_n$$

Lemma 171 gives us that everything up to u_{i-1} and u'_{i-1} coincide. We then get

$$[\![M]\!]_{q_i}(t_{\sigma(i)})...[\![M]\!]_{q_n}(t_{\sigma(n)})u_n = [\![M']\!]_{q'_i}(t_{\sigma'(i)})...[\![M']\!]_{q'_n}(t_{\sigma'(n)})u'_n$$

Since q'_i is not erasing, we can fix $t_{\sigma'(i)}$ such that $[\![M']\!]_{q'_i}(t_{\sigma'(i)}) \neq \varepsilon$. We call its first letter a. All non-ε productions of q_i must begin by a. This is only possible in an earliest if there exists $t^{\varepsilon}_{\sigma(i)}$ such that $[\![M]\!]_{q_i}(t^{\varepsilon}_{\sigma(i)}) = \varepsilon$. We now fix $t_{\sigma(i)} = t^{\varepsilon}_{\sigma(i)}$. If $u_i \neq \varepsilon$, its first letter is a. This is impossible in an earliest since it would mean $lcp(q_i u_i) \neq \varepsilon$. Hence $u_i = \varepsilon$ We can make the same reasoning for q_{i+1} and u_{i+1}, and so on all the way to q_{j-1} and u_{j-1}.

For point (B), we use point (A) to eliminate everything in front of q_k and $q'_{k'}$ by picking all $t^{\varepsilon}_{\sigma(l)}$ up to $k-1$ and all $t^{\varepsilon}_{\sigma'(l')}$ up to $k'-1$.

$$[\![M]\!]_{q_k}(t_{\sigma(k)})...[\![M]\!]_{q_n}(t_{\sigma(n)})u_n = [\![M']\!]_{q'_{k'}}(t_{\sigma'(k')})...[\![M']\!]_{q'_n}(t_{\sigma'(n)})u'_n$$

From Lemma 166, we know that $|[\![M]\!]_{q_k}(t_{\sigma(k)})| = |[\![M']\!]_{q'_{k'}}(t_{\sigma(k)})|$. We conclude that q_k and $q'_{k'}$ are equivalent.

For point (C), we take k' such that $\sigma(k) = \sigma'(k')$. We use (A) to erase everything but q_k, q_j, q'_i and $q'_{k'}$ by picking every $t^{\varepsilon}_{\sigma(l)}$ and $t^{\varepsilon}_{\sigma'(l')}$ except theirs.

$$[\![M]\!]_{q_k}(t_{\sigma(k)})[\![M]\!]_{q_j}(t_{\sigma(j)})...u_n = [\![M']\!]_{q'_i}(t_{\sigma'(i)})[\![M']\!]_{q'_{k'}}(t_{\sigma'(k')})...u'_n$$

Point (B) gives q_k is equivalent to $q'_{k'}$ and q_j is equivalent to q'_i. We get that $[\![M]\!]_{q_k}(t_{\sigma(k)})[\![M]\!]_{q_j}(t_{\sigma(j)}) = [\![M]\!]_{q_j}(t_{\sigma(j)})[\![M]\!]_{q_k}(t_{\sigma(k)})$. This means that the productions of q_k and q_j commute, which in Δ^* is equivalent to say they are words of same period. Therefore, q_j and q_k are periodic of same period.

This result allows us to resolve the first order different between two twin rules by flipping q_j with neighbouring periodic states of same period. We can iterate this method to solve all order differences.

Theorem 173. *For two twin rules of erase-ordered earliest* Ltws*,*

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)})u_n$$

$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)})u'_n$$

One can replace the rule of q to another rule of same subtree order as the rule of q' only by flipping neighbour states q_k and q_{k+1} of same period where $u_k = \varepsilon$.

We can use Lemma 172 to solve the leftmost difference: for i first index such that $\sigma(i) \neq \sigma'(i)$, and j such that $\sigma(i) = \sigma'(j)$, we have $u_i = ... = u_{j-1} = \varepsilon$ and $q_i, ..., q_j$ commute with each other. This means we can replace the first rule by:

$$q, f \to u_0 ... q_j(x_{\sigma(j)}) q_i(x_{\sigma(i)}) ... q_{j-1}(x_{\sigma(j-1)}) u_j ... u_n$$

where $q_j(x_{\sigma(j)})$ is to the left of $q_i(x_{\sigma(i)})...q_{j-1}(x_{\sigma(j-1)})$ without changing $[\![M]\!]_q$.

This solves the leftmost order difference: we can iterate this method until both rules have the same order.

Finally, we call Lemma 171 on the rules reordered by Theorem 173 to show that two twin rules use equivalent states and the same constant words:

Theorem 174. *For two twin rules of erase-ordered earliest* Ltw*s,*

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$$

$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma'(n)}) u'_n$$

$u_0 = u'_0, ..., u_n = u'_n$, *and for* k, k' *such that* $\sigma(k) = \sigma'(k')$, $[\![M]\!]_{q_k} = [\![M']\!]_{q'_{k'}}$.

7.5 Myhill-Nerode Theorem and Normal Form

In Section 7.3, we showed that Ltws can be made earliest. In Section 7.4, we first showed that all earliest Ltws can be made erase-ordered, then we made explicit what reorderings are possible between two rules of two equivalent states. In this section, we use these results to fix a reordering strategy. This will give us a new normal form, *ordered earliest* Ltws. We will show that each Ltw in equivalent to a unique minimal ordered earliest Ltw, whose size is at worst exponential.

We first use Theorem 173 to define a new normal form: ordered earliest Ltws.

Definition 175. *A* Ltw M *is said to be* ordered earliest *if it is earliest, and for each rule* $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$*:*

- *If* q_i *is erasing, then for any* $j > i$, q_j *is erasing.*

- *If* $u_i = \varepsilon$, *and* q_i *and* q_{i+1} *are periodic of same period,* $\sigma(i) < \sigma(i+1)$.

Note that this definition notably implies that any ordered earliest is erase-ordered earliest. On top of that, we impose that if two adjacent states are periodic of same period, and thus could be flipped, they are sorted by ascending subtree.

Lemma 176. *For* M *an earliest* Ltw*, one can make* M *ordered in polynomial time without changing the semantic of its states.*

We saw in Lemma 170 that one can push and sort erasing states. For this result, sorting periodic states is not more complicated. However, one must test first whether two states are periodic of same period. This can be done in polynomial time. One can prove that the productions of a Ltw state q form an algebraic language (described by a context-free grammar). Then, the problem of deciding if two algebraic languages are periodic of same period is known to be polynomial.

Our goal is now to show the existence of a unique minimal normal Ltw equivalent to any M. To this end, we first show that two equivalent Ltw*s* will use the same states: any $q \in Q$ has an equivalent $q' \in Q'$.

Lemma 177. *For two equivalent earliest* LTWs *M and M', for q state of M, there exist an equivalent state q' in M'.*

Proof. We start by the axioms: if $ax = u_0 q_0 v_0$ and $ax' = u'_0 q'_0 v'_0$, since M and M' are earliest, $u_0 = lcp([\![M]\!]) = lcp([\![M']\!]) = u'_0$. Then, $v_0 = lcs(q_0 v_0) = lcs(q'_0 v'_0) = v'_0$. We then get that q_0 and q'_0 are equivalent.

We can then call Theorem 174 to twin rules of equivalent states q, q' to get new equivalent pairs q_k, $q'_{k'}$ for $\sigma(k) = \sigma'(k')$. Since M is trimmed, this recursive calls will eventually reach all $q \in Q$ and pair them with an equivalent $q' \in Q'$. □

Since all equivalent earliest LTWs use the same states, they have the *minimal* amount of states when they don't have two redundant states q, q' such that $[\![M]\!]_q = [\![M]\!]_{q'}$. We show this characterises a *unique minimal normal form.*

Theorem 178. *For M a* LTW*, there exists a unique minimal ordered earliest* LTW *M' equivalent to M (up to state renaming).*

The existence of such a minimal ordered earliest LTW derives directly from Lemma 176. All we need to make an ordered earliest M' minimal is to merge its equivalent states together, which is always possible without changing $[\![M']\!]$.

The uniqueness derives from several properties we showed in this chapter. Imagine M and M' two equivalent minimal ordered earliest LTWs. The fact that they have equivalent states come from Lemma 177. Since both are minimal, neither have redundant state: each q of M is equivalent to exactly one q' of M' and vice-versa. From Theorem 174, we know that two equivalent states call equivalent states in their rules, with only the possibility of reordering periodic states. Since M and M' are ordered, twin rules also have same order.

7.6 Conclusion and Future Work

This chapter's goal was to solve the equivalence problem on linear tree-to-word transducers, by establishing a normal form and a Myhill-Nerode theorem on this class. To do so we naturally extended the notion of earliest transducers that already existed in sequential tree transducers [Laurence et al., 2011]. However it appeared that this was no longer enough to define a normal form: we studied all possible reorderings that could happen in an earliest LTW. We then used this knowledge to define a new normal form, that has both an output strategy (earliest) and an ordering strategy (ordered earliest), computable from any LTW in EXPTIME.

There are several ways to follow up on this result: one would be adapting the learning algorithm presented in [Laurence et al., 2014], accounting for the fact that we now also have to learn the order in which the images appear. It could also be relevant to note that in [Laurence et al., 2011], another algorithm decides equivalence in polynomial time, which is more efficient than computing the normal form. Such an algorithm would be an improvement over the actual randomized polynomial algorithm by [Seidl et al., 2015]. As far as Myhill-Nerode theorems go, the next step would be to consider all tree-to-word transducers. This problem is known to be difficult. Recently, [Seidl et al., 2015] gave a randomized polynomial algorithm to decide equivalence, but did not provide a Myhill-Nerode characterization.

Chapter 8

Equivalence of Linear Tree-to-Word Transducers is in Polynomial Time

Abstract. We show that the equivalence of linear top-down tree-to-word transducers is decidable in polynomial time. Linear tree-to-word transducers are non-copying but not necessarily order-preserving and can be used to express XML and other document transformations. The result is based on a partial normal form that provides a basic characterization of the languages produced by linear tree-to-word transducers.

Contents

8.1 Introduction **152**
8.2 Preliminaries **152**
8.3 Linear Tree-to-Word Transducers **153**
8.4 Partial Normal Form **155**
8.4.1 Eliminating Non-Earliest Quasi-Periodic States 156
8.4.2 Switching Periodic States 163
8.4.3 Testing Equivalence in Polynomial Time 164

8.1 Introduction

In Chapter 7, we proved the existence of an earliest ordered normal form for the class of linear tree-to-word transducers (Ltws). This normal form can be exponentially bigger than an equivalent Ltw, and its construction can be done in exponential time. This result came to extend the result of [Laurence et al., 2011] that proved the existence of an earliest normal form for the class of sequential (linear and order-preserving) tree-to-word transducers (Stws), that can be exponentially bigger than an equivalent Stw, and whose construction can be done in exponential time.

In this chapter, we wish prove the equivalence problem on Ltws to be in polynomial time. The equivalence of unrestricted tree-to-word transducers was a long standing open problem that was recently shown to be decidable [Seidl et al., 2015]. The algorithm by [Seidl et al., 2015] provides an co-randomized polynomial algorithm for linear transducers. We show that the equivalence of Ltws is decidable in polynomial time and provide a partial normal form.

To decide equivalence of Ltws, we start in Section 8.3 by extending the methods used for Stws, discussed in [Staworko et al., 2009]. The equivalence for these transducers is decidable in polynomial time [Staworko et al., 2009]. Two equivalent Ltws do not necessarily transform their trees in the same order. However, as seen in Chapter 7, the differences that can occur are quite specific. In their earliest form, two equivalent Ltws can transform subtrees in different orders only if they fulfill specific properties pertaining to the periodicity of the words they create. Computing this normal form is exponential in complexity as the number of states may increase exponentially. To avoid this size increase and obtain an equivalence algorithm in polynomial time, we do not compute these earliest transducers fully, but rather locally. This means we transform two Ltws with different orders to a *partial normal form* in polynomial time (see Section 8.4) where the order of their transformation of the different subtrees are the same. Ltws that transform the subtrees of the input in the same order can be reduced to sequential tree-to-word transducers as the input trees can be reordered according to the order in the transformation.

8.2 Preliminaries

We consider ranked trees on a finite signature F, and words on a finite alphabet Δ.

A *context-free grammar* (CFG) is defined as a tuple (Δ, N, S, P), where Δ is the alphabet of G, N is a finite set of *non-terminal symbols*, $S \in N$ is the initial non-terminal of G, P is a finite set of rules of form $A \to w$, where $A \in N$ and $w \in (\Delta \cup N)^*$. A CFG is deterministic if each non-terminal has at most one rule.

We define the language $L_G(A)$ of a non-terminal A recursively: if $A \to u_0A_1u_1...A_nu_n$ is a rule of P, with u_i words of Δ^* and A_i non-terminals of N, and w_i a word of $L_G(A_i)$, then $u_0w_1u_1...w_nu_n$ is a word of $L_G(A)$. We define the context-free language L_G of a context-free grammar G as $L_G(S)$.

A *straight-line program* (SLP) is a deterministic CFG that produces exactly one word. The word produced by an SLP (Δ, N, S, P) is called w_S.

We denote the *longest common prefix of all words* of a language L by $lcp(L)$. Its *longest common suffix* is $lcs(L)$.

A word u is said to be *periodic* of period w if w is the smallest word such that $u \in w^*$. A language L is said to be *periodic* of period w if w is the smallest word such that $L \subseteq w^*$.

A language L is *quasi-periodic* on the left (resp. on the right) of handle u and period w if w is the smallest word such that $L \subseteq uw^*$ (resp. if $L \subseteq w^*u$). A language is quasi-periodic if it is quasi-periodic on the right or left. If L is a singleton or empty, it is periodic of period ε. Iff L is periodic, it is quasi-periodic on the left and the right of handle ε. If L is quasi-periodic on the left (resp. right) then $lcp(L)$ (resp. $lcs(L)$) is the shortest word of L.

8.3 Linear Tree-to-Word Transducers

We use Ltws as they are defined in Chapter 7. For simplicity's sake, we only consider Ltws with non-empty domains and assume w.l.o.g. that no state q in an Ltw has an empty domain by eliminating transitions using states with empty domain.

Since a state's production L_q and their periodicty are of particular relevance to this Chapter, we extend the notion of periodic state to quasi-periodic: we call a state q *periodic* if L_q is periodic, and we call a state q *(quasi-)periodic* if L_q is (quasi-)periodic.

Note that a word u in a rule of an Ltw can be represented by an SLP without changing the semantics of the Ltw. Therefore a set of SLPs is added to the transducer and a word on the right-hand side of a rule is represented by an SLPs. The decidability of equivalence of Stws in polynomial time still holds true with the use of SLPs.

The results of this Chapter require SLP compression to avoid exponential blow-up. SLPs are used to prevent exponential blow-up in [Plandowski, 1995], where morphism equivalence on context-free languages is decided in polynomial time.

The equivalence problem for sequential tree-to-word transducer can be reduced to the morphism equivalence problem for context-free languages [Staworko et al., 2009]. This reduction relies on the fact that STWs transform their subtrees in the same order. As Ltws do not necessarily transform their subtrees in the same order the result cannot be applied on LTWs in general. However, if two LTWs transform their subtrees in the same order, then the same reduction can be applied. To formalize that two LTWs transform their subtrees in the same order we introduce the notion of state co-reachability.

Two states q_1 and q_2 of LTWs M_1, M_2, respectively, are co-reachable if there is an input tree such that the two states are assigned to the same node of the input tree in the translations of M_1, M_2, respectively.

Two LTWs are *same-ordered* if for each pair of co-reachable states q_1, q_2 and for each symbol $f \in F$, neither q_1 nor q_2 have a rule for f, or if $q_1, f \to u_0 q'_1(x_{\sigma_1(1)}) \dots q'_n(x_{\sigma_1(n)}) u_n$ and $q_2, f \to v_0 q''_1(x_{\sigma_2(1)}) \dots q''_n(x_{\sigma_2(n)}) v_n$ are rules of q_1 and q_2, then $\sigma_1 = \sigma_2$.

If two LTWs are same-ordered the input trees can be reordered according to the order in the transformations. Therefore for each LTW a tree-to-tree transducer is constructed that transforms the input tree according to the transformation in the LTW. Then all permutations σ in the LTWs are replaced by the identity. Thus the LTWs can be handled as STWs and therefore the equivalence is decidable in polynomial time [Staworko et al., 2009].

Theorem 179. *The equivalence of same-ordered* LTW*s is decidable in polynomial time.*

We remind that any LTW can be made *earliest* in exponential time. We also reminds that the construction of an earliest LTW referenced in Lemma 161 leaves the order unchanged, which means that it gives a *same-ordered* earliest LTW. However, we know from Theorem 173 and Theorem 174 that the only way for equivalent earliest and erase-ordered LTWs to not be same-ordered is to switch periodic states. This notably implies that two co-reachable states in equivalent earliest LTWs are necessarily equivalent.

However, building the earliest form of an LTW is in EXPTIME. To circumvent this difficulty, we will show that part of Theorem 173 still holds even on a *partial normal form*, where only quasi-periodic states are earliest and the longest common prefix of parts of rules $q(x)u$ with $L_q u$ being quasi-periodic is the empty word.

Theorem 180. *Let M and M' be two equivalent erase-ordered* LTW*s such that*

- all quasi-periodic states q are earliest, i.e. $lcp(q) = lcs(q) = \varepsilon$

- for each part $q(x)u$ of a rule where $L_q u$ is quasi-periodic, $lcp(L_q u) = \varepsilon$

Let q, q' be two co-reachable states in M, M', respectively and

$q, f \to u_0 q_1(x_{\sigma_1(1)}) \dots q_n(x_{\sigma_1(n)}) u_n$ and $q', f \to v_0 q'_1(x_{\sigma_2(1)}) \dots q'_n(x_{\sigma_2(n)}) v_n$

be two rules for q, q'. Then for $k < l$ such that $\sigma_1(k) = \sigma_2(l)$, all q_i, $k \le i \le l$, are periodic of the same period and all $u_j = \varepsilon$, $k \le j < l$.

Proof. Let M_e and M'_e be the equivalent earliest transducer of M and M', respectively, such that M and M_e as well as M' and M'_e are same-ordered (cf. Theorem 161).

Suppose there exists co-reachable (and thus equivalent) states q^e and q'^e in M_e and M'_e, respectively, with rules

$$q^e, f \to v_0 q^e_1(x_{\sigma(1)}) \dots q^e_n(x_{\sigma(n)})v_n,$$

$$q'^e, f \to v'_0 q'^e_1(x_{\sigma'(1)}) \dots q'^e_n(x_{\sigma(n)})v'_n$$

such that $\sigma \neq \sigma'$.

Let i be the first index such that $\sigma(i) \neq \sigma'(i)$. Following Theorem 173, we have j, j' such that $\sigma'(j') = \sigma(i)$ and $\sigma(j) = \sigma'(i)$ and all q^e_l, $i \leq l \leq j$ are periodic with the same period.

Let q and q' be the states in M and M', respectively, from which the co-reachable states q^e and q'^e were constructed with the earliest construction proposed by [Laurence et al., 2011]. From the earliest construction it follows that q and q' are co-reachable. Since the construction preserves the rule structure, we have:

$$q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)})u_n$$

$$q', f \to u'_0 q'_1(x_{\sigma'(1)}) \dots q'_n(x_{\sigma(n)})u'_n$$

The earliest construction gives us that for all $l \in \{1, \dots, n\}$, $[\![M_e]\!]_{q^e_l}(t) = v^{-1}u^{-1}[\![M]\!]_{q_l}(t)v$ for some $u, v \in \Delta^*$. This means that if q^e_l is periodic, then q_l is quasi periodic in its non-earliest form. The same is true for all q'_l.

However, the first property we supposed of M and M' implies that all those q_l and q'_l that are quasi-periodic are not only quasi periodic, but periodic. Consider a part of the rule $q_i(x_{\sigma(i)})u_i \dots q_j(x_{\sigma(j)})$ that is periodic in the earliest form and therefore quasi-periodic in the non-earliest form. The first condition gives us that $q_i, \dots, q_j$ are periodic. However, then the words $u_i, \dots, u_{j-1}$ are not necessarily empty. As the part $q_i(x_{\sigma(i)})u_i \dots q_j(x_{\sigma(j)})$ is quasi-periodic we know that each part $q_k(x_{\sigma(k)})u_k$, $i \leq k < j$ is quasi-periodic. Then the second condition of this theorem guarantees that the parts $q_k(x_{\sigma(k)})u_k$, $i \leq k < j$ are not only quasi-periodic, but periodic. From which it follows that the words $u_i, \dots, u_{j-1}$ are empty. As the part $q_i(x_{\sigma(i)})u_i \dots q_j(x_{\sigma(j)})$ is periodic and $u_i, \dots, u_{j-1}$ are empty we get that $q_i, \dots, q_j$ are periodic of the same period. The same holds true for states of a part of the rule $q'_i(x_{\sigma'(i)})u'_i \dots q'_j(x_{\sigma'(j)})$ that is periodic in the earliest form. □

8.4 Partial Normal Form

In this section we introduce a partial normal form for LTWs that does not suffer from the exponential blow-up of the earliest form. Inspired by Theorem 180, we wish to solve order differences by switching adjacent periodic states of the same period. Remember that the earliest form of a state q is constructed by removing the longest common prefix (suffix) of L_q to produce this prefix (suffix) earlier. It follows that all non-earliest states from which q can be constructed following the earliest form are quasi-periodic.

We show that building the earliest form of a quasi-periodic state or a part of a rule $q(x)u$ with $L_q u$ being quasi-periodic is in polynomial time. Therefore building the following partial normal form is in polynomial time.

Definition 181. *A linear tree-to-word transducer is in* partial normal form *if*

1. *all quasi-periodic states are earliest,*
2. *it is erase-ordered and*
3. *for each rule $q, f \rightarrow u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$ if $L_{q_i} u_i L_{q_{i+1}}$ is quasi-periodic then $q_i(x_{\sigma(i)}) u_i q_{i+1}(x_{\sigma(i+1)})$ is earliest and $\sigma(i) < \sigma(i+1)$.*

8.4.1 Eliminating Non-Earliest Quasi-Periodic States

In this part, we show a polynomial time algorithm to build an earliest form of a quasi-periodic state. From which an equivalent LTW can be constructed in polynomial time such that any quasi-periodic state is earliest, i.e. $lcp(L_q) = lcs(L_q) = \varepsilon$. Additionally, we show that the presented algorithm can be adjusted to test if a state is quasi-periodic in polynomial time.

As quasi-periodicity on the left and on the right are symmetric properties we only consider quasi-periodic states of the form uw^* (quasi-periodic on the left). The proofs in the case w^*u are symmetric and therefore omitted here. In the end of this section we shortly discuss the introduced algorithms for the symmetric case w^*u.

To build the earliest form of a quasi-periodic state we use the property that each state accessible from a quasi-periodic state is as well quasi-periodic. However, the periods can be shifted as the following example shows.

Example 182. *Consider states q, q_1 and q_2 with rules $q, f \rightarrow aq_1(x_1)c$, $q_1, f \rightarrow aaq_2(x_1)ab$, $q_2, f \rightarrow q_2(x_1)abc$, $q_2, g \rightarrow abc$. State q accepts trees of the form $f^n(g)$, $n \geq 2$, and produces the language $aaa(abc)^n$, i.e. q is quasi-periodic of period abc. State q_1 accepts trees of the form $f^n(g)$, $n \geq 1$, and produces the language $aa(abc)^n ab$, i.e. q_1 is quasi-periodic of period cab. State q_2 accepts trees of the form $f^n(g)$, $n \geq 0$ and produces the language $(abc)^{n+1}$, i.e. q_2 is (quasi-)periodic of period abc.*

We introduce two definitions to measure the shift of periods. We denote by $\rho_n[u]$ the *from right-to-left shifted word of u of shift n*, $n \leq |u|$, i.e. $\rho_n[u] = u'^{-1}uu'$ where u' is the prefix of u of size n. If $n \geq |u|$ then $\rho_n[u] = \rho_m[u]$ with $m = n \mod |u|$.

For two quasi-periodic states q_1, q_2 of period $u = u_1 u_2$ and $u' = u_2 u_1$, respectively, we denote the *shift in their period* by $s(q_1, q_2) = |u_1|$.

The size of the periods of a quasi-periodic state and the states accessible from this state can be computed from the size of the shortest words of the languages produced by these states.

Lemma 183. *If q is quasi-periodic on the left with period w, and q' accessible from q, then q' is quasi-periodic with period ε or a shift of w. Moreover we can calculate the shift $s(q, q')$ in polynomial time.*

Proof. This is done as an iterative proof with the following elementary step: If $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$, and q is quasi-periodic on the left with handle u and period w, then for all i between 1 and n, q_i is quasi-periodic with period ε or a shift of w.

We pick v_j a smallest word produced by state q_j. We then have that for all $t \in dom(q_i)$, $u_0 v_1 ... u_{i-1} [\![M]\!]_{q_i}(t) u_i ... v_n u_n \in L_q$. If we call $u_l = u_0 v_0 ... u_{i-1}$ and $u_r = u_i ... v_n u_n$, we obtain that $L_{q_i} \subseteq u_l^{-1} L_q u_r^{-1}$. Since $L_q \subseteq uw^*$, we can say $L_{q_i} \subseteq u_l^{-1}(uw^*)u_r^{-1}$. It is a classical result of regular languages that $(uw^*)u_r^{-1}$ is either empty, a singleton, or a quasi-periodic language of period $u_r w u_r^{-1}$. By further removing a prefix to this language the period does not change. Hence, we get that $u_l^{-1}(uw^*)u_r^{-1}$ is also either empty, a singleton, or a quasi-periodic language of period $u_r w u_r^{-1}$. This means that q_i is quasi-periodic, of period ε, or $u_r w u_r^{-1}$, which is a shift of q. The size of u_r can easily be computed from the sizes of the minimal productions of states q_j. We build the CFG for L_{q_j}. Then, finding the smallest production of q_j and their size is finding the smallest word of L_{q_j} and their size, which is a polynomial problem on CFG.

To show that the shifts of the periods can be calculated in polynomial time we show that shifts are additive in nature: If q_1 has period w, q_1 and q_2 are of shifted period, and q_2 and q_3 are of shifted period, then q_1 and q_3 are of shifted period, and $s(q_1, q_3) \equiv s(q_1, q_2) + s(q_2, q_3) \pmod{|w|}$.

If q_1 is of period w, then q_2 is of period $w_2 = w' w w'^{-1}$, where w' is the suffix of w of size $s(q_1, q_2)$. If q_2 is of period w_2, then q_2 is of period $w_3 = w'' w_2 w''^{-1}$, where w'' is the suffix of w_2 of size $s(q_2, q_3)$. We then have that $w_3 = (w'' w') w (w'' w')^{-1}$, where $(w'' w')$ is of size $s(q_1, q_2) + s(q_2, q_3)$.

We can compute the shift of the period of each state accessible from q rule by rule using the additive property of the shifts we proved above. □

We now use these shifts to build, for a state q in M that is quasi-periodic on the left, a transducer M^q equivalent to M where each occurrence of q is replaced by its equivalent earliest form, i.e. a periodic state and the corresponding prefix.

Algorithm 184. *Let q be a state in M that is quasi-periodic on the left. M^q starts with the same states, axiom, and rules as M.*

- *For each state p accessible from q, we add a copy p^e to M^q.*
- *For each rule $p, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)}) u_n$ in M with p accessible from q, we add a rule $p^e, f \to u_p q_1^e(x_{\sigma(1)}) q_2^e(x_{\sigma(2)}) \dots q_n^e(x_{\sigma(n)})$ with $u_p = \rho_{s(q,p)}\left[lcp(p)^{-1} u_0 lcp(q_1) \dots lcp(q_n) u_n\right]$ in M^q.*
- *We delete state q in M^q and replace any occurrence of $q(x)$ in a rule or the axiom of M^q by $lcp(q)q^e(x)$.*

Note that $lcp(p)^{-1}u_0 lcp(q_1) \dots lcp(q_n)u_n$ is equivalent to deleting the prefix of size $|lcp(p)|$ from the word $u_0 lcp(q_1) \dots lcp(q_n)u_n$.

Intuitively, to build the earliest form of a state q that is quasi-periodic on the left we need to push all words and all longest common prefixes of states on the right-hand side of a rule of q to the left. Pushing a word to the left through a state needs to shift the language produced by this state. We explain the algorithm in detail on state q from Example 182.

Example 185. *Remember that q produces the language $aaa(abc)^n, n \geq 2$ and q_1, q_2 accessible from q produce languages $aa(abc)^n ab, n \geq 1$ and $(abc)^{n+1}, n \geq 0$, respectively. Therefore $lcp(q) = aaaabcabc$, $lcp(q_1) = aaabcab$ and $lcp(q_2) = abc$. We start with state q. As there is only one rule for q the longest common prefix of q and the longest common prefix of this rule are the same and therefore eliminated.*

$$\begin{aligned} q^e, f &\to \rho_{s(q,q)}[lcp(q)^{-1} a lcp(q_1) c] q_1^e(x_1) \\ &\to \rho_{s(q,q)}[(aaaabcabc)^{-1} aaaabcabc] q_1^e(x_1) \\ &\to q_1^e(x_1) \end{aligned}$$

As there is only one rule for q_1 the argumentation is the same and we get $q_1^e, f \to q_2^e$. For the rule q_2, f we calculate the longest common prefix of the right-hand side $lcp(q_2)abc = abcabc$ that is larger than the longest common prefix of q_2. Therefore we need to calculate the shift $s(q, q_2) = s(q, q_1) + s(q_1, q_2) = |c| + |ab| = 3$ as q_1 is accessible from q in rule q, f and q_2 is accessible from q_1 in rule q_1, f. This leads to the following rule.

$$\begin{aligned} q_2^e, f &\to \rho_{s(q,q_2)}[lcp(q_2)^{-1} lcp(q_2) abc] q_2^e(x_1) \\ &\to \rho_3[(abc)^{-1} abcabc] q_2^e(x_1) \\ &\to abc q_2^e(x_1) \end{aligned}$$

As the longest common prefix of q_2 is the same as the longest common prefix of the right-hand side of rule q_2, g we get $q_2^e, g \to \varepsilon$. The axiom of M^q is $lcp(q)q^e(x_1) = aaaabcabcq^e(x_1)$.

Lemma 186. *Let M be an* Ltw *and q be a state in M that is quasi-periodic on the left. Let M^q be constructed by Algorithm 184 and p^e be a state in M^q accessible from q^e. Then M and M^q are equivalent and p^e is earliest.*

Proof. To show that M and M^q are equivalent we show that $lcp(q)[\![M^q]\!]_{q^e}(t) = [\![M]\!]_q(t)$, for all $t \in dom(q)$. To show that $lcp(q)[\![M^q]\!]_{q^e}(t) = [\![M]\!]_q(t)$ we show that, for all states p accessible from q and all $t \in dom(p)$, $[\![M^q]\!]_{p^e}(t)$ and $\rho_{s(q,p)}\,[lcp(p)^{-1}[\![M]\!]_p(t)]$ are equivalent as then

$$\begin{aligned} lcp(q)[\![M^q]\!]_{q^e}(t) &= lcp(q)\rho_{s(q,q)}\left[lcp(q)^{-1}[\![M]\!]_q(t)\right] \\ &= lcp(q)lcp(q)^{-1}[\![M]\!]_q(t) \\ &= [\![M]\!]_q(t). \end{aligned}$$

To show that $[\![M^q]\!]_{p^e}(t) = \rho_{s(q,p)}\,[lcp(p)^{-1}[\![M]\!]_p(t)]$ for all p accessible from q and all $t = (s_1,\dots,s_n) \in dom(p)$, we prove that $[\![M^q]\!]_{p^e}(t)$ is of the same period and of the same size as $\rho_{s(q,p)}\,[lcp(p)^{-1}[\![M]\!]_p(t)]$. From Lemma 183 we know that all p accessible from q are quasi-periodic and therefore $lcp(p)^{-1}[\![M]\!]_p(t)$ is periodic. Hence, if $[\![M^q]\!]_{p^e}(t)$ and $\rho_{s(q,p)}\,[lcp(p)^{-1}[\![M]\!]_p(t)]$ are of the same period and of the same size then they are equivalent.

To show that $[\![M^q]\!]_{p^e}(t)$ and $\rho_{s(q,p)}\,[lcp(p)^{-1}[\![M]\!]_p(t)]$ have the same size, for all $t \in dom(p)$, we show that $|[\![M^q]\!]_{p^e}(t)| = |[\![M]\!]_p(t)| - |lcp(p)|$. The proof is by induction on the input tree. For an input tree t with no subtrees we have

$$\begin{aligned} |[\![M^q]\!]_{p^e}(t)| &= |\rho_{s(q,p)}[lcp(p)^{-1}u_0\dots u_n]| \\ &= |u_0| + \dots + |u_n| - |lcp(p)| \\ &= |[\![M]\!]_p(t)| - |lcp(p)| \\ &= |\rho_{s(q,p)}\left[lcp(p)^{-1}[\![M]\!]_p(t)\right]|. \end{aligned}$$

Thus, the base casef holds. Consider an input tree $t = f(s_1,\dots,s_n) \in dom(p)$. Then $[\![M^q]\!]_{p^e}(t)$ is of size $|u_p| + |[\![M^q]\!]_{q_1^e}(s_{\sigma(1)})| + \dots + |[\![M^q]\!]_{q_n^e}(s_{\sigma(n)})|$ with $|u_p| = |\rho_{s(q,p)}[lcp(p)^{-1}u_0 lcp(q_1)\dots lcp(q_n)u_n]|$. Since shifting a word preserves its length, we have $|u_p| = |u_0| + |lcp(q_1)| + \dots + |u_n| - |lcp(p)|$. Thus, we have to show that

$$|u_p| + |[\![M^q]\!]_{q_1^e}(s_{\sigma(1)})| + \dots + |[\![M^q]\!]_{q_n^e}(s_{\sigma(n)})| = |\rho_{s(q,p)}\left[lcp(p)^{-1}[\![M]\!]_p(t)\right]|.$$

By induction we have $|[\![M^q]\!]_{q_i^e}(s_{\sigma(i)})| = |[\![M]\!]_{q_i}(s_{\sigma(i)})| - |lcp(q_i)|$. Thus, we have

$$\begin{aligned} |u_p|+&|[\![M^q]\!]_{q_1^e}(s_{\sigma(1)})| + \dots + |[\![M^q]\!]_{q_n^e}(s_{\sigma(n)})| \\ &= |u_0| + |lcp(q_1)| + \dots + |lcp(q_n)| + |u_n| - |lcp(p)| \\ &\qquad + |[\![M]\!]_{q_1}(s_{\sigma(1)})| - |lcp(q_1)| + \dots + |[\![M]\!]_{q_n}(s_{\sigma(n)})| - |lcp(q_n)| \\ &= |u_0| + |[\![M]\!]_{q_1}(s_{\sigma(1)})| + \dots + |[\![M]\!]_{q_n}(s_{\sigma(n)})| + |u_n| - |lcp(p)| \\ &= |[\![M]\!]_p(t)| - |lcp(p)| \\ &= |\rho_{s(q,p)}\left[lcp(p)^{-1}[\![M]\!]_p(t)\right]|. \end{aligned}$$

To show that $[\![M^q]\!]_{p^e}(t)$ and $\rho_{s(q,p)}\,[lcp(p)^{-1}[\![M]\!]_p(t)]$ have the same period, for all $t \in dom(p)$, we show that $[\![M^q]\!]_{p^e}(t) \in u^*$ and $\rho_{s(q,p)}\,[lcp(p)^{-1}[\![M]\!]_p(t)] \in u^*$ where $L_q \subseteq wu^*$. From Lemma 183 it follows that $\rho_{s(q,p)}\,[lcp(p)^{-1}[\![M]\!]_p(t)] \in u^*$. To proof that $[\![M^q]\!]_{p^e}(t) \in u^*$

is by induction on the input tree. For an input tree t with no subtrees we have $[\![M^q]\!]_{p^e}(t) = \rho_{s(q,p)}\left[lcp(p)^{-1}u_0 \dots u_n\right]$. From Lemma 183 we know that L_p is quasi-periodic of period $u'u''$ where $u = u''u'$ and u' is of size $s(q,p)$. Thus, $lcp(p)^{-1}u_0 \dots u_n \in (u'u'')^*$ and therefore $\rho_{s(q,p)}\left[lcp(p)^{-1}u_0 \dots u_n\right] \in (u'u'')^* = u^*$. Hence, the base case holds. Consider an input tree $t = f(s_1, \dots, s_n) \in dom(p)$. Then we have $[\![M^q]\!]_{p^e}(t) = \rho_{s(q,p)}\left[lcp(p)^{-1}u_0 lcp(q_1) \dots lcp(q_n)u_n\right] [\![M^q]\!]_{q_1^e}(s_{\sigma(1)}) \dots [\![M^q]\!]_{q_n^e}(s_{\sigma(n)})$. By induction, $[\![M^q]\!]_{q_i^e}(s_{\sigma(i)}) \in u^*$. With the same argumentation as in the base case $lcp(p)^{-1}u_0 lcp(q_1) \dots lcp(q_n)u_n \in (u'u'')^*$ with $u = u''u'$ and u' is of size $s(q,p)$. Thus, $\rho_{s(q,p)}\left[lcp(p)^{-1}u_0 lcp(q_1) \dots lcp(q_n)u_n\right] \in (u''u')^* = u^*$ and therefore we get $[\![M^q]\!]_{p^e}(t) \in u^*$. □

To replace all quasi-periodic states by their equivalent earliest form we need to know which states are quasi-periodic. Algorithm 184 can be modified to test an arbitrary state for quasi-periodicity on the left in polynomial time. The only difference to Algorithm 184 is that we do not know how to compute $lcp(p)$ in polynomial time and $s(q,p)$ does not exist. We therefore substitute $lcp(p)$ by some smallest word of L_p and we define a mock-shift $s'(q,p)$ as follows

- $s'(q,q) = 0$ for all q,
- if $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)})u_n$, we say $s'(q,q_i) = |u_i w_{q_{i+1}} \dots w_{q_n} u_n|$, where w_q is a shortest word of L_q,
- if $s'(q_1,q_2) = n$ and $s'(q_2,q_3) = m$ then $s'(q_1,q_3) = n + m$.

If several definitions of $s'(q,p)$ exist, we use the smallest. If p is accessible from a quasi-periodic q, then $s'(q,p) = s(q,p)$.

Algorithm 187. *Let $M = (F, \Delta, Q, ax, rul)$ be an* Ltw *and q be a state in M. We build an* Ltw *T^q as follows.*

- *For each state p accessible from q, we add a copy p^e to T^q.*
- *The axiom is $w_q q^e(x)$ where w_q is a shortest word of L_q.*
- *For each rule $p, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)})u_n$ in M with p accessible from q, we add a rule*

$$p^e, f \to u_p q_1^e(x_{\sigma(1)}) q_2^e(x_{\sigma(2)}) \dots q_n^e(x_{\sigma(n)})$$

  in T^q, where u_p is constructed as follows.
  - *We define $u = u_0 w_1 \dots w_n u_n$, where w_i is a shortest word of L_{q_i}.*
  - *Then we remove from u its prefix of size $|w'|$, where w' is a shortest word of L_p. We obtain a word u'.*
  - *Finally, we set $u_p = \rho_{s'(q,p)}[u']$.*

As the construction of Algorithms 184 and 187 are the same if the state q is quasi-periodic, $[\![M]\!]_q$ and $[\![T^q]\!]$ are equivalent if q is quasi-periodic. Moreover, q is quasi-periodic if $[\![M]\!]_q$ and $[\![T^q]\!]$ are equivalent.

Lemma 188. *Let q be a state of an* Ltw *M and T^q be constructed by Algorithm 187. Then M and T^q are same-ordered and q is quasi-periodic on the left if and only if $[\![M]\!]_q = [\![T^q]\!]$ and q^e is periodic.*

Proof. We show that q is quasi-periodic on the left if and only if $[\![M]\!]_q = [\![T^q]\!]$ and q^e is periodic. If q is quasi-periodic on the left the transformation in Algorithm 187 is the same as in Algorithm 184. Therefore $[\![M]\!]_q = [\![T^q]\!]$ and q^e is periodic.

If $[\![M]\!]_q = [\![T^q]\!]$ and q^e is periodic, then $[\![M]\!]_q$ is quasi-periodic as $[\![T^q]\!] = w_q[\![T^q]\!]_{q^e}$ with w_q a shortest word of L_q.

M and T^q are same-ordered as the order of the rules in T^q is the same as in M by construction. □

As M and T^q are same-ordered we can test the equivalence in polynomial time, cf. Theorem 179. Moreover testing a CFG for periodicity is in polynomial time and therefore testing a state for quasi-periodicity is in polynomial time.

Algorithm 187 can be applied to a part $q(x)u$ of a rule to test $L_q u$ for quasi-periodicity on the left. In this case for each rule $q, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)})u_n$ a rule $\hat{q}, f \to u_0 q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)})u_n u$ is added to M and each occurrence of the part $q(x)u$ in a rule of M is replaced by $\hat{q}(x)$. We then apply the above algorithm to $\hat{q}$ and test $[\![M]\!]_{\hat{q}}$ and $[\![T^{\hat{q}}]\!]$ for equivalence and $\hat{q}^e$ for periodicity.

Example 189. *Let q be a state with the rules $q, f \to bcaq(x_1)$, $q, g \to \varepsilon$. Thus, q transforms trees of the form $f^n(g)$, $n \geq 0$ to $(bca)^n$. We use Algorithm 187 to test $L_q bc$ for quasi-periodicity on the left. As explained above we introduce a state $\hat{q}$ with the rules $\hat{q}, f \to bca\hat{q}(x_1)$, $\hat{q}, g \to bc$. We now apply Algorithm 187 on $\hat{q}$. We build $T^{\hat{q}} = \{\{f, g\}, \{a, b, c\}, \{\hat{q}^e\}, ax, \delta\}$ as follows. The axiom ax is $bc\hat{q}^e(x_0)$ as the shortest word of $L_{\hat{q}}$ is bc. For the rule $\hat{q}, f$ we build $u = bcabc$ as bc is the shortest word of $\hat{q}$. Then we obtain $u' = abc$ and $u_{\hat{q}} = \rho_{s'(\hat{q},\hat{q})}[abc] = abc$. Thus we get $\hat{q}^e, f \to abc\hat{q}^e(x_1)$. For the rule $\hat{q}, g$ we build $u = bc$ and obtain $u' = \varepsilon$ as the shortest word of $\hat{q}$ is bc. Thus we get $\hat{q}^e, g \to \varepsilon$.*

$T^{\hat{q}}$ transforms trees of the form $f^n(g)$ to $bc(abc)^n$ and $\hat{q}$ transforms trees of the form $f^n(g)$ to $(bca)^n bc$. Thus, they are equivalent. Additionally $\hat{q}^e$ is periodic with period abc. It follows that $L_{q_1}bc$ is quasi-periodic.

We introduced algorithms to test states for quasi-periodicity on the left and to build the earliest form for such states. These two algorithms can be adapted for states that are quasi-periodic on the right. There are two main differences. First, as the handle is on the right the shortest word of a language L that is quasi-periodic on the right is $lcs(L)$. Second, instead of pushing words through a periodic language to the left we need to push words through a periodic language to the right.

Hence, we can test each state q of an LTW M for quasi-periodicity on the left and right. If the state is quasi-periodic we replace q by its earliest form. Algorithm 184 and 187 run in polynomial time if SLPs are used. This is crucial as the shortest word of a CFG can be of exponential size. However, the operations that are needed in the algorithms, namely constructing the shortest word of a CFG and removing the prefix or suffix of a word, are in polynomial time using SLPs, cf. [Lohrey, 2014].

Theorem 190. *Let M be an* LTW. *Then an equivalent* LTW *M' where all quasi-periodic states are earliest can be constructed in polynomial time.*

Proof. This proof works by induction. We first show that if $M = (F, \Delta, Q, ax, rul)$ has n, $n \geqslant 1$ quasi-periodic states that are non-earliest, then we can build in polynomial time an equivalent LTW M' with $n-1$ non-earliest quasi-periodic states. Using Algorithm 187 we choose q as a non-earliest quasi-periodic state of Q. We apply Algorithm 184 on state q and get M^q, whose set of state is of form $Q \sqcup Q^e \backslash \{q\}$, where Q^e is the set of states p^e with p accessible from q that are created by Algorithm 184. According to Lemma 186 all the states of Q^e are periodic. This means that the non-earliest quasi-periodic states of M^q are all in $Q\backslash\{q\}$. Since Q has n non-earliest quasi-periodic states, including q, M^q has $n-1$.

Now we can build M_1 equivalent to M with $n-1$ non-earliest quasi-periodic states, then M_2 equivalent to M_1 (hence to M) with $n-2$ non-earliest quasi-periodic states, and so on. Finally we get M_n equivalent to M with no non-earliest quasi-periodic state. Each step is in polynomial time and the number n is smaller than the number of states in M. For each occurence of a state on the right-hand side of a rule there is at most one new state needed in the construction. Therefore the size increase of the transducer is only polynomial. To avoid the construction of equivalent states q should be considered before q' if q' is accessible from q. If q is accessible from q' and q' is accessible from q then q is considered first if there is a acyclic way from the axiom to q that contains q'.

In the above proof we assumed that Algorithm 187 and 184 run in polynomial time. In both algorithms it is crucial that SLPs are used to represent the shortest words of the languages produced by the states of a transducer as these can be of exponential size. Instead of these uncompressed words nonterminals representing these words as SLPs are inserted in the transducers. All operations that are needed in the algorithms, namely constructing a SLP for the shortest word of an CFG, concatenation of SLPs, shifting the word produced by an SLP and removing the prefix or suffix of an SLP are in polynomial time [Lohrey, 2014].

□

8.4.2 Switching Periodic States

In this part we obtain the partial normal form by ordering periodic states of an erase-ordered transducer where all quasi-periodic states are earliest. Ordering means that if the order of the subtrees in the translation can differ, we choose the one similar to the input, i.e. if $q(x_3)q'(x_1)$ and $q'(x_1)q(x_3)$ are equivalent, we choose the second order. We already showed how we can build a transducer where each quasi-periodic state is earliest and therefore periodic. However, we need to make parts of rules earliest such that periodic states can be switched as the following example shows.

Example 191. *Consider the rule $q, h \to q_1(x_2)bq_2(x_1)$ where q_1, q_2 have the rules $q_1, f \to bcabcaq_1(x)$, $q_1, g \to \varepsilon$, $q_2, f \to cabq_2(x)$, $q_2, g \to \varepsilon$. States q_1 and q_2 are earliest and periodic but not of the same period as a subword is produced in between. We replace the non-earliest and quasi-periodic part $q_1(x_2)b$ by their earliest form. This leads to $q, h \to bq_1^e(x_2)q_2(x_1)$ with $q_1^e, f \to cabcabq_1^e(x)$, $q_1^e, g \to \varepsilon$. Hence, q_1^e and q_2 are earliest and periodic of the same period and can be switched in the rule.*

To build the earliest form of a quasi-periodic part of a rule $q(x)u$ each occurrence of this part is replaced by a state $\hat{q}(x)$ and for each rule $q, f \to u_0q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)})u_n$ a rule $\hat{q}, f \to u_0q_1(x_{\sigma(1)}) \dots q_n(x_{\sigma(n)})u_nu$ is added. Then we apply Algorithm 184 on $\hat{q}$ to replace $\hat{q}$ and therefore $q(x)u$ by their earliest form. Iteratively this leads to the following theorem.

Theorem 192. *For each* Ltw *M where all quasi-periodic states are earliest we can build in polynomial time an equivalent* Ltw *M' such that each part $q(x)u$ of a rule in M where L_qu is quasi-periodic is earliest.*

In Theorem 180 we showed that order differences in equivalent erase-ordered Ltws where all quasi-periodic states are earliest and all parts of rules $q(x)u$ are earliest are caused by adjacent periodic states. As these states are periodic of the same period and no words are produced in between these states can be reordered without changing the semantics of the Ltws.

Lemma 193. *Let M be an* Ltw *such that*
- *M is erase-ordered,*
- *all quasi-periodic states in M are earliest and*
- *each $q_i(x_{\sigma(i)})u_i$ in a rule of M that is quasi-periodic is earliest.*

Then we can reorder adjacent periodic states $q_i(x_{\sigma(i)})q_{i+1}(x_{\sigma(i+1)})$ of the same period in the rules of M such that $\sigma(i) < \sigma(j)$ in polynomial time. The reordering does not change the transformation of M.

We showed before how to construct a transducer with the preconditions needed in Lemma 193 in polynomial time. Note that replacing a quasi-periodic state by its earliest form can break the erase-ordered property. Thus we need to replace all quasi-periodic states by its earliest form *before* building the erase-ordered form of a transducer. Then Lemma 193 is the last step to obtain the partial normal form for an LTW.

Theorem 194. *For each* LTW *we can construct an equivalent* LTW *that is in partial normal form in polynomial time.*

8.4.3 Testing Equivalence in Polynomial Time

It remains to show that the equivalence problem of LTWs in partial normal form is decidable in polynomial time. The key idea is that two equivalent LTWs in partial normal form are same-ordered.

Consider two equivalent LTWs M_1, M_2 where all quasi-periodic states and all parts of rules $q(x)u$ with $L_q u$ is quasi-periodic are earliest. In Theorem 180 we showed if the orders σ_1, σ_2 of two co-reachable states q_1, q_2 of M_1, M_2, respectively, for the same input differ then the states causing this order differences are periodic with the same period. The partial normal form solves this order differences such that the transducers are same-ordered.

Lemma 195. *If M and M' are equivalent and in partial normal form then they are same-ordered.*

As the equivalence of same-ordered LTWs is decidable in polynomial time (cf. Theorem 179) we conclude the following.

Corollary 196. *The equivalence problem for* LTW*s in partial normal form is decidable in polynomial time.*

To summarize, the following steps run in polynomial time and transform a LTW M into its partial normal form.

1. Test each state for quasi-periodicity. If it is quasi-periodic replace the state by its earliest form.
2. Build the equivalent erase-ordered transducer.
3. Test each part $q(x)u$ in each rule from right to left for quasi-periodicity on the left. If it is quasi-periodic replace the part by its earliest form.
4. Order adjacent periodic states of the same period according to the input order.

This leads to our main theorem.

Theorem 197. *The equivalence of* LTW*s is decidable in polynomial time.*

The equivalence problem for linear tree-to-word transducers can be decided in polynomial time. To prove this we used a reduction to the equivalence problem between sequential transducers [Laurence et al., 2011], or more exactly, to an extension of this result to same-ordered transducers. This reduction hinges on two points. First, we showed that the only structural differences between two equivalent earliest linear transducers are caused by periodic languages which are interchangeable. The structural characteristic of periodic languages has been used in the normalization of STWs [Laurence et al., 2011]. Second, we showed that if building a fully earliest transducer is potentially exponential, our reduction only requires quasi-periodic states to be earliest, which can be done in polynomial time. The use of the equivalence problem for morphisms on a CFG [Plandowski, 1995] and of properties on straight-line programs [Lohrey, 2014] is essential here as it was in [Laurence et al., 2011, Laurence et al., 2014]. This leads to further research questions, starting with generalization of this result to all tree-to-words transducers. Furthermore, is it possible that these techniques can be used to decrease the complexity of some problems in other classes of transducer classes, such as top-down tree-to-tree transducers, where the equivalence problem is known to be between EXPTIME-Hard and NEXPTIME?

Chapter 9
Conclusion

The common thread running throughout this thesis was the desire to study learning problems, and the related equivalence and normalization problems, for tree transducers. The existing results in this domain were scarce, and subject to multiple limitations, three of which we addressed in our thesis: learning results on transducers schema restrictions, on transducers with lookaheads, and on transducers with concatenation operations in the output. We wish to present some perspectives of potential future work. We will consider the three restrictions we studied, as well as the question of data tree transducers.

For the problem of schema restriction, we learn DTops with regular domain thanks to a Myhill-Nerode theorem on the DTopI$_{\text{reg}}$ class. It is unlikely for those results to extend to all kinds of schemas restrictions, as these restrictions can be much more complex than what was considered here. To extend learning problems to schema-restricted languages or transformations will probably require a more general study, that does not seek to make the machine work as closely to its schema as possible, but rather allows to learn an object when restricted to an incomplete subset of examples, by guessing how the transformation would perform outside the schema restriction. This problem of learning with an incomplete access to examples has been studied for the Angluin learning model [Angluin and Slonim, 1994], but remains mostly open. We remind the example mentioned in the introduction as a fitting starting point: we try to learn a learning regular word language L when restricted to a schema S that describe an algebraic language $[\![S]\!]$. Note that if Dfas can be learned, context-free grammars cannot, which means we cannot learn a normal form on $L \cap [\![S]\!]$.

For transducers with lookahead, we showed a learning algorithm for the case on words, but did not address the extension of this result or even the existence of a normal form to tree transducers with lookahead. It is already known that the minimal lookahead method [Reutenauer and Schützenberger, 1991] cannot apply to the tree case, as some transformations do not possess a unique minimal lookahead. However, we can design a minimal equivalence relation on subtrees such that replacing one with another can only generate a finite difference. This relation is not sufficient to build a lookahead for its transformation in the general case, nor is it obvious how to refine it to do so. Normalizing this refinement could lead to a normal form on tree transducers with lookahead.

For concatenation, whether the results we present for Ltws can be extended to all tree-to-word transducers is linked to the more fundamental problem of finding a normal form on a formalism that allows both concatenation and copying in its output. This case has only been studied in two-way word transducers with origin information [Bojańczyk, 2014]. This origin information, however, is difficult to learn if not provided. We propose a possible first step towards the resolution of this problem, a finite concatenation of subsequential word transducers. Imagine a pair of transducers $N = (M_1, M_2)$ that defines a word transformation: $[\![N]\!](u) = [\![M_1]\!](u) \cdot [\![M_2]\!](u)$. This class's equivalence is decidable, as it is strictly weaker than the two-way formalism. An interesting problem would be to find a normal form on this class, and potentially a learning algorithm. Another solution to find normal forms on tree transducers with concatenation would be to restrict what concatenations are allowed in the output. For example, one could restrict macro tree transducers to forbid them to output the transformation of a subtree below another transformation of the same subtree, effectively making those transducers linear in each branch of the output.

Transducers on data trees are an extension we did not address in this thesis. Several approaches exist to do so. Symbolic transducers on words [Veanes et al., 2012] and on trees [Veanes and Bjørner, 2011] propose results on transducers that can read and transform data. The main constraint on the result presented in trees is that equivalence is proven decidable for the case where data in the input can only be used once in the output. We also mention the class of data word automata with registers, that possesses a minimal normal form [Manuel et al., 2013], and could possibly be extended into tree automata and transducers with data registers on which such results could extend to provide a learnable class of data transducers. Another interesting open problem is the extent of data interactions than can be performed in a transducer: the MSO logic with data constraints we discussed in the CoLiS [Boiret et al., 2015] seems to lead to the possibility of a transducer that can compare and potentially combine data in the same arity. However, such a formalism has yet to be developed.

The open problems and perspectives we present here are mainly motivated by the extension we can guess for existing results, but many more can and should be found from concrete applications. The theory of automata and transducers provides a solid framework to find results on verification problems for increasingly complex tree transformations. However, it is worth noting that whenever transducer methods are used to model concrete applications, they need to be adapted to encompass their specificities. One such example would be the CoLiS project that drives us to develop transducers for unranked data trees in order to verify properties on installation scripts. Every use of transducer methods creates new classes with new functionalities, on which to extend and adapt previously existing results and methods.

Bibliography

[Abiteboul, 1997] Abiteboul, S. (1997). Querying semi-structured data. In *Proceedings of the 6th International Conference on Database Theory*, ICDT '97, pages 1–18, London, UK, UK. Springer-Verlag. (Cited on page 1.)

[Abiteboul et al., 2011] Abiteboul, S., Manolescu, I., Rigaux, P., Rousset, M., and Senellart, P. (2011). *Web Data Management*. Cambridge University Press. (Cited on page 1.)

[Allemang and Hendler, 2008] Allemang, D. and Hendler, J. (2008). *Semantic web for the working ontologist : effective modeling in RDF, RDFS and OWL*. Morgan Kaufmann Publishers/Elsevier, Amsterdam ; Boston. (Cited on page 1.)

[Alur and Cerný, 2010] Alur, R. and Cerný, P. (2010). Expressiveness of streaming string transducers. In Lodaya, K. and Mahajan, M., editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. (Cited on page 23.)

[Alur and Cerný, 2011] Alur, R. and Cerný, P. (2011). Streaming transducers for algorithmic verification of single-pass list-processing programs. In Ball, T. and Sagiv, M., editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 599–610. ACM. (Cited on page 23.)

[Angluin, 1987] Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106. (Cited on pages 6, 51 and 99.)

[Angluin and Slonim, 1994] Angluin, D. and Slonim, D. K. (1994). Randomly fallible teachers: Learning monotone DNF with an incomplete membership oracle. *Machine Learning*, 14(1):7–26. (Cited on page 167.)

[Barker, 2016] Barker, D. (2016). *Web Content Management: Systems, Features, and Best Practices*. O'Reilly Media. (Cited on page 2.)

[Bassett, 2015] Bassett, L. (2015). *Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON*. O'Reilly Media. (Cited on page 1.)

[Benedikt et al., 2013] Benedikt, M., Engelfriet, J., and Maneth, S. (2013). Determinacy and rewriting of top-down and MSO tree transformations. In Chatterjee, K. and Sgall, J., editors, *Mathematical Foundations of*

Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings, volume 8087 of *Lecture Notes in Computer Science*, pages 146–158. Springer. (Cited on page 10.)

[Benedikt and Koch, 2007] Benedikt, M. and Koch, C. (2007). XPath leashed. *ACM computing surveys.* (Cited on page 3.)

[Benzaken et al., 2003] Benzaken, V., Castagna, G., and Frisch, A. (2003). CDuce: an XML-centric general-purpose language. *ACM SIGPLAN Notices*, 38(9):51–63. (Cited on page 4.)

[Benzaken et al., 2013] Benzaken, V., Castagna, G., Nguyen, K., and Siméon, J. (2013). Static and dynamic semantics of NoSQL languages. In Giacobazzi, R., Cousot, R., Giacobazzi, R., and Cousot, R., editors, *POPL*, pages 101–114. ACM. (Cited on page 1.)

[Berstel, 1979] Berstel, J. (1979). *Transductions and Context-Free Languages.* Teubner Studienbucher. (Cited on pages 8, 11 and 126.)

[Bex et al., 2008] Bex, G. J., Gelade, W., Neven, F., and Vansummeren, S. (2008). Learning deterministic regular expressions for the inference of schemas from XML data. In *17-th International Conference on World Wide Web.* (Cited on page 51.)

[Bex et al., 2002] Bex, G. J., Maneth, S., and Neven, F. (2002). A formal model for an expressive fragment of XSLT. *Information Systems*, 27:21–39. (Cited on page 49.)

[Beyer et al., 2011] Beyer, K. S., Ercegovac, V., Gemulla, R., Eltabakh, M., and Balmin, A. (2011). Jaql: A scripting language for large scale semistructured data analysis. vldb. (Cited on page 2.)

[Boiret, 2016] Boiret, A. (2016). Normal form on linear tree-to-word transducers. In Dediu, A., Janousek, J., Martín-Vide, C., and Truthe, B., editors, *Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*, volume 9618 of *Lecture Notes in Computer Science*, pages 439–451. Springer. (Cited on page 20.)

[Boiret et al., 2014] Boiret, A., Hugot, V., Niehren, J., and Treinen, R. (2014). Deterministic automata for unordered trees. In Peron, A. and Piazza, C., editors, *Proceedings Fifth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014, Verona, Italy, September 10-12, 2014.*, volume 161 of *EPTCS*, pages 189–202. (Cited on page 20.)

[Boiret et al., 2015] Boiret, A., Hugot, V., Niehren, J., and Treinen, R. (2015). Logics for unordered trees with data constraints on siblings. In Dediu, A. H., Formenti, E., Martín-Vide, C., and Truthe, B., editors, *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume 8977 of *Lecture Notes in Computer Science*, pages 175–187. Springer. (Cited on pages 20 and 168.)

[Boiret et al., 2016a] Boiret, A., Hugot, V., Niehren, J., and Treinen, R. (2016a). Automata for Unordered Trees. *Information and Computation*. (Cited on page 20.)

[Boiret et al., 2012] Boiret, A., Lemay, A., and Niehren, J. (2012). Learning Rational Functions. In *16th International Conference on Developments of Language Theory*, pages 273–283, Taipee, Taiwan, Province Of China. (Cited on page 20.)

[Boiret et al., 2016b] Boiret, A., Lemay, A., and Niehren, J. (2016b). Learning Top-Down Tree Transducers with Regular Domain Inspection. In *International Conference on Grammatical Inference 2016*, Delft, Netherlands. (Cited on page 19.)

[Boiret and Palenta, 2016] Boiret, A. and Palenta, R. (2016). Deciding equivalence of linear tree-to-word transducers in polynomial time. In Brlek, S. and Reutenauer, C., editors, *Developments in Language Theory - 20th International Conference, DLT 2016, Montréal, Canada, July 25-28, 2016, Proceedings*, volume 9840 of *Lecture Notes in Computer Science*, pages 355–367. Springer. (Cited on page 20.)

[Bojańczyk, 2014] Bojańczyk, M. (2014). Transducers with origin information. In Esparza, J., Fraigniaud, P., Husfeldt, T., and Koutsoupias, E., editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 26–37. Springer. (Cited on pages 23 and 168.)

[Bojanczyk et al., 2011] Bojanczyk, M., David, C., Muscholl, A., Schwentick, T., and Segoufin, L. (2011). Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27+. (Cited on page 25.)

[Bonifati et al., 2015] Bonifati, A., Ciucanu, R., and Lemay, A. (2015). Interactive path query specification on graph databases. In Alonso, G., Geerts, F., Popa, L., Barceló, P., Teubner, J., Ugarte, M., den Bussche, J. V., and Paredaens, J., editors, *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 505–508. OpenProceedings.org. (Cited on page 51.)

[Bonifati et al., 2016] Bonifati, A., Ciucanu, R., and Staworko, S. (2016). Learning join queries from user examples. *ACM Trans. Database Syst.*, 40(4):24. (Cited on page 51.)

[Bray et al., 2008] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergea, F. (2008). Extensible markup language (XML) 1.0 (fifth edition). W3C Recommendation. (Cited on page 1.)

[Carme et al., 2007] Carme, J., Gilleron, R., Lemay, A., and Niehren, J. (2007). Interactive learning of node selecting tree transducers. *Machine Learning*, 66(1):33–67. (Cited on pages 37, 50, 51 and 99.)

[Carme et al., 2004] Carme, J., Lemay, A., and Niehren, J. (2004). Learning node selecting tree transducer from completely annotated examples. In *7th International Colloquium on Grammatical Inference*, volume 3264 of *Lecture Notes in Artificial Intelligence*, pages 91–102. Springer Verlag. (Cited on page 24.)

[Castagna et al., 2015] Castagna, G., Im, H., Nguyen, K., and Benzaken, V. (2015). A core calculus for xquery 3.0 - combining navigational and pattern matching approaches. In Vitek, J., editor, *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9032 of *Lecture Notes in Computer Science*, pages 232–256. Springer. (Cited on page 4.)

[Choffru, 1978] Choffru, C. (1978). *Contribution a l'etude de quelques familles remarquables de fonctions rationnelles*. PhD thesis, Universite de Paris VII. (Cited on page 68.)

[Choffrut, 1979] Choffrut, C. (1979). A generalization of ginsburg and rose's characterisation of g-s-m mappings. In *ICALP 79*, number 71 in Lecture Notes in Computer Science, pages 88–103. Springer Verlag. (Cited on page 128.)

[Choffrut, 2003] Choffrut, C. (2003). Minimizing subsequential transducers: a survey. *Theoretical Computer Science*, 292(1):131–143. (Cited on pages 8, 50, 51, 68, 128 and 141.)

[Comon et al., 2007] Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., and Tommasi, M. (2007). Tree automata techniques and applications. Available online since 1997: `http://tata.gforge.inria.fr`. (Cited on pages 33, 34 and 36.)

[Courcelle, 1994] Courcelle, B. (1994). Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53–75. (Cited on pages 21 and 22.)

[D'Antoni and Veanes, 2014] D'Antoni, L. and Veanes, M. (2014). Minimization of symbolic automata. In Jagannathan, S. and Sewell, P., editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 541–554. ACM. (Cited on page 25.)

[de la Higuera, 2010] de la Higuera, C. (2010). *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press. (Cited on page 6.)

[Doan et al., 2001] Doan, A., Domingos, P., and Halevy, A. (2001). Reconciling schemas of disparate data sources: A Machine-Learning approach. In *Proceedings of the ACM SIGMOD Conference*, pages 509–520. (Cited on page 51.)

[Elgot and Mezei, 1965] Elgot, C. C. and Mezei, G. (1965). On relations defined by generalized finite automata. *IBM J. of Res. and Dev.*, 9:88–101. (Cited on pages 17, 99, 126 and 128.)

[Engelfriet, 1975] Engelfriet, J. (1975). Bottom-up and top-down tree transformations — a comparison. *Math. Systems Theory*, 9:198–231. (Cited on pages 24 and 55.)

[Engelfriet, 1977] Engelfriet, J. (1977). Top-down tree transducers with regular look-ahead. *Mathematical System Theory*, 10:198–231. (Cited on pages 10, 57, 99 and 136.)

[Engelfriet and Courcelle, 2012] Engelfriet, J. and Courcelle, B. (2012). *Graph Structure and Monadic Second-order Logic*. Number 138 in Encyclopedia of Mathematics and its Applications. Cambridge University Press. (Cited on page 21.)

[Engelfriet and Hoogeboom, 2001] Engelfriet, J. and Hoogeboom, H. J. (2001). MSO definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2:216–254. (Cited on page 22.)

[Engelfriet and Maneth, 2003] Engelfriet, J. and Maneth, S. (2003). Macro tree translations of linear size increase are MSO definable. *SIAM Journal on Computing*, 4(32):950–1006. (Cited on pages 9, 24 and 49.)

[Engelfriet and Maneth, 2005] Engelfriet, J. and Maneth, S. (2005). The equivalence problem for deterministic MSO tree transducers is decidable.

In Ramanujam, R. and Sen, S., editors, *Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science – FSTTCS'2005*, volume 3821 of *LNCS*, pages 495–504. Springer-Verlag. (Cited on pages 9, 22, 24 and 49.)

[Engelfriet et al., 2009] Engelfriet, J., Maneth, S., and Seidl, H. (2009). Deciding equivalence of top-down XML transformations in polynomial time. *Journal of Computer and System Science*, 75(5):271–286. (Cited on pages 9, 13, 15, 46, 49, 50, 52, 53, 55, 59, 66, 68, 70, 71, 80, 99 and 141.)

[Engelfriet et al., 2016] Engelfriet, J., Maneth, S., and Seidl, H. (2016). Look-ahead removal for total deterministic top-down tree transducers. *Theor. Comput. Sci.*, 616:18–58. (Cited on page 14.)

[Engelfriet and Vogler, 1985] Engelfriet, J. and Vogler, H. (1985). Macro tree transducer. *Journal of Computer and System Science*, 31:71–146. (Cited on pages 8, 11, 49 and 138.)

[Ésik, 1980] Ésik, Z. (1980). Decidability results concerning tree transducers I. *Acta Cybernetica*, 5:1–20. (Cited on pages 9 and 49.)

[Filiot and Reynier, 2014] Filiot, E. and Reynier, P. (2014). On streaming string transducers and HDT0L systems. *CoRR*, abs/1412.0537. (Cited on page 14.)

[Friese, 2011] Friese, S. (2011). *On Normalization and Type Checking for Tree Transducers*. PhD thesis, Technische Universität München, München. (Cited on page 24.)

[Friese et al., 2010] Friese, S., Seidl, H., and Maneth, S. (2010). Minimization of deterministic Bottom-Up tree transducers. In Gao, Y., Lu, H., Seki, S., and Yu, S., editors, *Developments in Language Theory, 14th International Conference DLT 2010*, volume 6224 of *Lecture Notes in Computer Science*, pages 185–196. Springer Verlag. (Cited on page 24.)

[Fülöp and Maletti, 2013] Fülöp, Z. and Maletti, A. (2013). Composition closure of linear extended top-down tree transducers. *CoRR*, abs/1301.1514. (Cited on page 10.)

[Gold, 1967] Gold, E. M. (1967). Language identification in the limit. *Inform. Control*, 10:447–474. (Cited on page 6.)

[Gold, 1978] Gold, E. M. (1978). Complexity of automaton identification from given data. *Inform. Control*, 37:302–320. (Cited on pages 7 and 48.)

[Graehl et al., 2008] Graehl, J., Knight, K., and May, J. (2008). Training tree transducers. *Computational Linguistics*, 34(3):391–427. (Cited on page 51.)

[Hopcroft and Ullman, 1979] Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley. (Cited on page 29.)

[Hosoya, 2010] Hosoya, H. (2010). *Foundations of XML Processing: The Tree-Automata Approach.* Cambridge University Press. (Cited on page 3.)

[Janssen et al., 2007] Janssen, W., Korlyukov, A., and Van den Bussche, J. (2007). On the tree-transformation power of XSLT. *Acta Inf.*, 43(6):371–393. (Cited on page 49.)

[Kay, 2001] Kay, M. (2001). *XSLT: Programmer's Reference.* Programmer to programmer. Wiley. (Cited on page 3.)

[Kepser, 2004] Kepser, S. (2004). A simple proof for the Turing-Completeness of XSLT and XQuery. In *Extreme Markup Languages®*. (Cited on pages 7 and 49.)

[Knuth, 1986] Knuth, D. E. (1986). *The TeXbook.* Addison-Wesley Professional. (Cited on page 1.)

[Labath and Niehren, 2015] Labath, P. and Niehren, J. (2015). A uniform programmning language for implementing XML standards. In Italiano, G. F., Margaria-Steffen, T., Pokorný, J., Quisquater, J., and Wattenhofer, R., editors, *SOFSEM 2015: Theory and Practice of Computer Science - 41st International Conference on Current Trends in Theory and Practice of Computer Science, Pec pod Snvevzkou, Czech Republic, January 24-29, 2015. Proceedings*, volume 8939 of *Lecture Notes in Computer Science*, pages 543–554. Springer. (Cited on pages 4 and 55.)

[Laurence et al., 2011] Laurence, G., Lemay, A., Niehren, J., Staworko, S., and Tommasi, M. (2011). Normalization of sequential Top-Down Tree-to-Word transducers. In Dediu, A. H., Inenaga, S., Vide, C. M., Dediu, A. H., Inenaga, S., and Vide, C. M., editors, *LATA*, volume 6638 of *Lecture Notes in Computer Science*, pages 354–365. Springer. (Cited on pages 12, 14, 138, 139, 140, 141, 142, 143, 149, 150, 152, 155, 164 and 165.)

[Laurence et al., 2014] Laurence, G., Lemay, A., Niehren, J., Staworko, S., and Tommasi, M. (2014). Learning sequential tree-to-word transducers. In Dediu, A. H., Martín-Vide, C., Sierra-Rodríguez, J. L., and Truthe, B., editors, *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, volume 8370 of *Lecture Notes in Computer Science*, pages 490–502. Springer. (Cited on pages 12, 14, 19, 138, 139, 149 and 165.)

[Lemay et al., 2010] Lemay, A., Maneth, S., and Niehren, J. (2010). A Learning Algorithm for Top-Down XML Transformations. In Acm, editor, *29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 285–296, Indianapolis, United States. ACM Press. (Cited on pages 9, 13, 15, 16, 19, 24, 37, 46, 50, 59, 79, 102 and 139.)

[Lohrey, 2014] Lohrey, M. (2014). *The Compressed Word Problem for Groups*. Springer. (Cited on pages 162 and 165.)

[Maletti et al., 2009] Maletti, A., Graehl, J., Hopkins, M., and Knight, K. (2009). The power of extended top-down tree transducers. *SIAM J. Comput.*, 39(2):410–430. (Cited on page 10.)

[Maneth, 2015] Maneth, S. (2015). A survey on decidable equivalence problems for tree transducers. *Int. J. Found. Comput. Sci.*, 26(8):1069–1100. (Cited on pages 10 and 11.)

[Maneth et al., 2005] Maneth, S., Berlea, A., Perst, T., and Seidl, H. (2005). XML type checking with macro tree transducers. In *24th ACM Symposium on Principles of Database Systems*, pages 283–294, New York, NY, USA. ACM-Press. (Cited on pages 8, 12, 14, 24, 49, 54 and 55.)

[Maneth and Busatto, 2004] Maneth, S. and Busatto, G. (2004). Tree transducers and tree compressions. In Walukiewicz, I., editor, *Foundations of Software Science and Computation Structures - FOSSACS'04*, volume 2987 of *LNCS*, pages 363–377, Barcelona, Spain. Springer-Verlag. (Cited on page 54.)

[Maneth and Neven, 1999] Maneth, S. and Neven, F. (1999). Structured document transformations based on XSL. In Connor, R. C. H., Mendelzon, A. O., Connor, R. C. H., and Mendelzon, A. O., editors, *DBPL*, volume 1949 of *Lecture Notes in Computer Science*, pages 80–98. Springer. (Cited on pages 4, 14, 49, 54 and 55.)

[Manuel et al., 2013] Manuel, A., Muscholl, A., and Puppis, G. (2013). Walking on data words. In Bulatov, A. A. and Shur, A. M., editors, *Computer Science - Theory and Applications - 8th International Computer Science Symposium in Russia, CSR 2013, Ekaterinburg, Russia, June 25-29, 2013. Proceedings*, volume 7913 of *Lecture Notes in Computer Science*, pages 64–75. Springer. (Cited on page 168.)

[Martens et al., 2006] Martens, W., Neven, F., Schwentick, T., and Bex, G. J. (2006). Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems*, 31(3):770–813. (Cited on page 3.)

[Marx, 2005] Marx, M. (2005). Conditional XPath. *ACM Transactions on Database Systems*, 30(4):929–959. (Cited on page 3.)

[Mohri, 2000] Mohri, M. (2000). Minimization algorithms for sequential transducers. *Theor. Comput. Sci.*, 234(1-2):177–201. (Cited on page 51.)

[Morishima et al., 2004] Morishima, A., Kitagawa, H., and Matsumoto, A. (2004). A machine learning approach to rapid development of XML mapping queries. In Özsoyoglu, Z. M., Zdonik, S. B., Özsoyoglu, Z. M., and Zdonik, S. B., editors, *ICDE*, pages 276–287. IEEE Computer Society. (Cited on page 51.)

[Nerode, 1958] Nerode, A. (1958). Linear automaton transformation. In *Proc. American Mathematical Society*, volume 9, pages 541–544. (Cited on pages 7, 30 and 64.)

[Neumann, 2000] Neumann, A. (2000). *Parsing and Quering XML Documents in SML*. PhD thesis, Universität Trier. (Cited on page 4.)

[Niehren et al., 2013] Niehren, J., Champavère, J., Lemay, A., and Gilleron, R. (2013). Query induction with schema-guided pruning strategies. *Journal of Machine Learning Research*, 14(1):927–964. (Cited on pages 13, 37 and 51.)

[Niese, 2003] Niese, O. (2003). *An integrated approach to testing complex systems*. Phd thesis, Universität Dortmund, Germany. (Cited on page 51.)

[Oncina and Garcia, 1992] Oncina, J. and Garcia, P. (1992). Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis*, pages 49–61. (Cited on pages 7, 36, 50 and 84.)

[Oncina and García, 1993] Oncina, J. and García, P. (1993). Inference of recognizable tree sets. Technical report, Departamento de Sistemas Informáticos y Computación, Universidad de Alicante. (Cited on page 37.)

[Oncina et al., 1993] Oncina, J., Garcia, P., and Vidal, E. (1993). Learning subsequential transducers for pattern recognition and interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:448–458. (Cited on pages 8, 37, 49, 51, 64 and 139.)

[Oncina and Varo, 1996] Oncina, J. and Varo, M. A. (1996). Using domain information during the learning of a subsequential transducer. In *ICGI 1996*, volume 1147 of *Lecture Notes in Artificial Intelligence*, pages 313–325. Springer Verlag. (Cited on pages 13, 126, 128 and 135.)

[Onder and Bayram, 2006] Onder, R. and Bayram, Z. (2006). XSLT version 2.0 is Turing-Complete: A purely transformation based proof. In Ibarra, O. H., Yen, H. C., Ibarra, O. H., and Yen, H. C., editors, *CIAA*, volume 4094 of *Lecture Notes in Computer Science*, pages 275–276. Springer. (Cited on pages 7 and 49.)

[Pezoa et al., 2016] Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., and Vrgoc, D. (2016). Foundations of JSON schema. In Bourdeau, J., Hendler, J., Nkambou, R., Horrocks, I., and Zhao, B. Y., editors, *Proceedings of the 25th International Conference on World Wide Web, WWW 2016, Montreal, Canada, April 11 - 15, 2016*, pages 263–273. ACM. (Cited on page 3.)

[Pilgrim, 2010] Pilgrim, M. (2010). *HTML5: Up and Running: Dive into the Future of Web Development*. O'Reilly Media. (Cited on page 1.)

[Plandowski, 1995] Plandowski, W. (1995). *The complexity of the morphism equivalence problem for context-free languages*. PhD thesis, Warsaw University. Department of Informatics, Mathematics, and Mechanics. (Cited on pages 12, 18, 153 and 165.)

[Popa et al., 2002] Popa, L., Velegrakis, Y., Miller, R. J., Hernández, M. A., and Fagin, R. (2002). Translating web data. In *VLDB*, pages 598–609. Morgan Kaufmann. (Cited on page 51.)

[Powell, 2006] Powell, G. (2006). *Beginning Xml Databases*. Wiley India Pvt. Limited. (Cited on page 1.)

[Powers, 2003] Powers, S. (2003). *Practical RDF*. O'Reilly Media. (Cited on page 1.)

[Reutenauer and Schützenberger, 1991] Reutenauer, C. and Schützenberger, M. P. (1991). Minimalization of rational word functions. *SIAM Journal on Computing*, 20:669–685. (Cited on pages 11, 14, 17, 18, 126, 129, 130 and 167.)

[Robinson et al., 2013] Robinson, I., Webber, J., and Eifrem, E. (2013). *Graph Databases*. O'Reilly Media. (Cited on page 1.)

[Sebastian and Niehren, 2016] Sebastian, T. and Niehren, J. (2016). Projection for Nested Word Automata Speeds up XPath Evaluation on XML Streams. In *International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, Harrachov, Czech Republic. (Cited on page 4.)

[Seidl et al., 2015] Seidl, H., Maneth, S., and Kemper, G. (2015). Equivalence of deterministic top-down tree-to-string transducers is decidable. In Guruswami, V., editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 943–962. IEEE Computer Society. (Cited on pages 9, 11, 14, 49, 54, 138, 150 and 152.)

[Siegel and Retter, 2014] Siegel, E. and Retter, A. (2014). *A NoSQL Document Database and Application Platform.* O'Reilly Media. (Cited on page 2.)

[Staworko et al., 2009] Staworko, S., Laurence, G., Lemay, A., and Niehren, J. (2009). Equivalence of nested word to word transducers. In *17th International Symposium on Fundamentals of Computer Theory*, volume 5699 of *Lecture Notes in Computer Science*, pages 310–322. Springer Verlag. (Cited on pages 152, 153 and 154.)

[Staworko and Wieczorek, 2015] Staworko, S. and Wieczorek, P. (2015). Characterizing XML twig queries with examples. In Arenas, M. and Ugarte, M., editors, *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium*, volume 31 of *LIPIcs*, pages 144–160. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. (Cited on page 51.)

[van der Vlist, 2002] van der Vlist, E. (2002). *XML Schema: The W3C's Object-Oriented Descriptions for XML.* O'Reilly Media. (Cited on page 2.)

[van der Vlist, 2003] van der Vlist, E. (2003). *RELAX NG.* O'Reilly Media. (Cited on page 2.)

[van der Vlist, 2007] van der Vlist, E. (2007). *Schematron.* O'Reilly Media. (Cited on page 2.)

[Veanes and Bjørner, 2011] Veanes, M. and Bjørner, N. (2011). Symbolic tree transducers. In Clarke, E. M., Virbitskaite, I., and Voronkov, A., editors, *Perspectives of Systems Informatics - 8th International Andrei Ershov Memorial Conference, PSI 2011, Novosibirsk, Russia, June 27-July 1, 2011, Revised Selected Papers*, volume 7162 of *Lecture Notes in Computer Science*, pages 377–393. Springer. (Cited on pages 25 and 168.)

[Veanes et al., 2012] Veanes, M., Hooimeijer, P., Livshits, B., Molnar, D., and Bjørner, N. (2012). Symbolic finite state transducers: algorithms and applications. In Field, J. and Hicks, M., editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 137–150. ACM. (Cited on pages 25 and 168.)

[Walsh, 2010] Walsh, N. (2010). *DocBook 5: The Definitive Guide.* O'Reilly Media. (Cited on page 1.)