Machine Learning Techniques for Semistructured Data

Aurélien Lemay

Université de Lille

Centre de Recherche en Informatique, Signal et Automatique de Lille (CRIStAL)

Habilitation à Diriger des Recherches

Spécialité "Informatique"

Techniques d'Apprentissage Automatique pour les Données Semistructurées

par
Aurélien Lemay

Soutenue le 16 Novembre 2018 devant le jury composé de :

M.	**BLOCKEEL Hendrik**	Professeur, Université de Leuven Professeur Associé, Université de Leiden	Rapporteur
M.	**FERNAU Henning**	Professeur, Université de Trier	Rapporteur
M.	**MALETTI Andreas**	Professeur, Université de Leipzig	Rapporteur
M.	**NIEHREN Joachim**	Directeur de Recherche, INRIA, Lille	Garant
M.	**TREINEN Ralf**	Professeur, Université Paris Diderot	Président du Jury

Machine Learning Techniques for Semistructured Data

Infer Queries and Transformations with Grammatical Inference Tools

Aurélien Lemay

Work Presented for the Habilitation Thesis

Table of Content

1. Table of Contents

1 Introduction

1. Databases on the Web

Much of today's data is accessible on the Web or in the cloud. This includes personal data (photos, music, films), social network data (friends and personal data), commercial data (selling offers, product presentations), medical data (on illnesses and medications), cultural data (museum expositions, cinema programs or music concerts), and common knowledge as in Wikipedia and so on. The sheer amount of data available is tremendous and in a permanent evolution.

The development of the Web comes alongside with a revolution of data management systems for collecting, storing and accessing all kinds of data. Some important changes can be highlighted:

- **The frontier between data producers and data consumers is completely blurred**, as everybody can contribute in many different ways: by adding articles on Wikipedia, by posting on a forum, by blogging or by participating in crowd sourcing for instance. This means that data comes in many ways and from a wide diversity of actors, ranging from average users to database experts.
- **As data comes from many sources, it also comes in many formats** from highly structured to unstructured ones (plain text), with many possibilities in-between of partially structured data. Databases thus have to represent and analyse data of heterogeneous nature.
- **This heterogeneity makes data difficult to access**. To deal with this, keyword search has offered an alternative to database queries. Thereby anybody can access the tremendous amount of data on the web in a convenient manner, or provide his own data so that it can be found by others.
- **The creation of data and its access has become highly dynamic.** While early web pages were static and produced entirely at creation time, modern web sites are typically built dynamically, often providing a convenient access to a database. A web shop for instance builds dynamically a web page on demand when a user wants information on a product. This dynamicity is also present on the web page itself through various Javascript code. This allows the web page to directly query - and possibly update - an underlying database.
- **The pure size of accessible data today has grown way beyond what could be stored in a single relational database**. Indeed, some systems have to store data on billions of individuals. Therefore, the distributed storage of data has become essential. In some situations, this size makes traditional relational database techniques unsuitable as they simply can not efficiently answer queries on databases of this volume.

The revolution of data management is still in progress, leading to new business models such as web stores like Amazon, new interaction models such as social networks, or new data collection models (Wikipedia for instance). However, database queries remain at the heart of most services offered on the Web. Queries are sometimes directly visible in form-based services that allow for instance to find a travel ticket, or to find a product in an e-commerce website. Most often however, they are hidden in various other Web services.

- Most modern websites are built using a **Content Management System** (CMS). They are systems that build webpages dynamically from contents stored in a relational database. The navigation in such a website generates a sequence of queries whose results allow to build webpages.

- **Keyword search engines**, like Google, search for the most probable answers using statistical methods and also database queries in order to access a semantic to that concept [GMM03]. For instance, when searching for Mozart with Google, a database query is then used to access information about his symphonies and his birth date.

- **Social networks** are implemented on top of distributed databases of the NoSQL family. These databases represent friendship relations but also all other kinds of personal information in a highly distributed manner. Those data are then queried to produce recommendations to users of the social network, for instance when a user searches for the friends of friends [Bar03, Jac08]

- **Crowdsourcing** systems such as Wikipedia/DBpedia [LIJ+15] fill up databases with all kinds of knowledge which, beside texts or pictures, contain linked data in RDF format. This data can then be queried by keyword search engines or other Web services [ADK+15, PPP+12].

- Web **information extraction** techniques allow to gather data from Web pages in order to offer new services based on data they harvest [Bae03, BHM04]. This requires to query Web pages written in HTML based on XML queries.

Database formats and query languages are evolving in parallel with the revolution of data management on the Web to allow to store and manage the new kind of data involved. When in the past, data tended to be considered either structured (as in a relational database) or unstructured (like plain text), it is now acknowledged that a whole spectrum of structuration levels exist. This diversity in structuration of data is usually designated by the term **semi-structured data**, a terminology introduced by Abiteboul [Abi97, ABS00]. This new paradigm revolves around generic relational data

structures such as trees or graphs. Those formats may allow a complete liberty in what kind of data they can store or how it is organised. In parallel, new adapted query languages have evolved.

Several data formats have emerged to follow this trend, the most popular ones being certainly the following.

- **XML** [MPSM+04] has became the de-facto standard for exchange of data on the Web, in particular for Web services and Web document publication. It allows to use a generic tree structure to store data. This genericity allowed a whole family of tools to exist, from schema languages such as DTD [MPSM+04], XML schema [xml] or Relax NG [rel], query languages (XPath 1.0 [CD99], XPath 2.0 [xpaa] and XPath 3.0 [xpab] most notably), or transformation languages (XSLT [Cla99] or XQuery [RSC+10] for instance).
- The greatest success of XML is certainly **HTML**, the language of Web pages. In its present version HTML 5.0 [Hic12] and associated with CSS stylesheet [css], it allows to define the most elaborated presentations of web pages on web browsers. As it is based on XML, it inherits from the whole set of tools that XML provides. In particular, data extraction can be done via XPath for instance.
- The **JSON** format [Cro14] is initially a storage format for Javascript data. It has therefore been designed to store and exchange data of web scripts in a convenient manner. Through the **AJAX** (Asynchronous Javascript and XML [Mah06]) architecture, it has now become an important element of dynamic web pages. It has also become one of the main standart of the **NoSQL** paradigm [Cro09, SF13], used in particular in many distributed databases. Its tree structure makes it perfectly adapted for distributed usages, as well as allowing very fast accesses and treatments, such as map-reduce techniques.
- The **semantic web** project aims to represent a maximum of data using knowledge graphs, based on the **RDF** format [rdf]. This format describes edges of a graph using triples of the form *subject - predicate - object*. It can be stored and/or exchanged via an XML description, but other serialisation formats also exist (Turtle, N3, ...). RDF databases are usually paired with an ontology that allows to infer new relations from existing one, much in a deductive way. It also allows to specify the semantics of data, with the possibility to share concepts among different databases. Queries are typically made using the **SPARQL** language [spa].

Alongside those different formats, there is also a **large variety of structuration levels**. Indeed, in the semi-structured data paradigm, one can find data with a high level of structuration where every piece of data is clearly semantically identified within a schema. However, it may happen that this

schema is underspecified, or that its structuration level does not cover every content (some data is left unstructured). Sometimes data is structured, but its structure does not carry semantic values. For instance in a Web page, HTML tags carry mostly syntactic information. It may also happen that the schema is completely unknown, is only implicit, or even that it evolves over time.

This diversity of data formats and query languages as well as the potential underspecification of schemas make the task of query design more complex than for relational databases where the SQL format dominates and where databases always have a clear schema. To circumvent these problems, **machine learning can provide solutions**. It provides tools that allow to automatically design queries by studying data and by using simple interactions with an end-user. The typical way to do this is by using methods of statistical machine learning. This has allowed to obtain nice results, with the advantage of having a great robustness with respect to noisy data for instance. However, statistical machine learning allows to obtain only approximations of the query wanted by the user. Our proposition in this field is to use symbolic machine learning instead. Under some well-designed scenarios that we study, this allows to obtain exact learning of queries.

2. Queries, Transformations and Learning

Semi-structured data is usually organised in relational structures over a finite alphabet, but with data values (strings, numbers...) from an infinite set. Which kind of relational structures are considered depends on the data format that is chosen, and on possible restrictions that are imposed from the schema.

As in classical relational databases, the management of semi-structured data is usually based on **logical queries** on the relational structure. A logical query is a function that maps relational structures to relations. Those relations can then possibly be transformed into a data tree or a data graph. As an example, an XPath query, as specified by the W3C standart [CD99], maps data trees in the XML data model to sets of nodes of the tree. XPath queries are also at the core of the XSLT [Cla99] or XQuery [RSC+10] formalisms, which allow to transform XML data trees into arbitrary relational structures, possibly in other XML data trees.

Logic is the most natural approach to define query languages for relational structures [AHV95]. Most typically, one considers queries defined in **first-order logic** (FO) or fragments thereof such as conjunctive queries. Trakhtenbrot's theorem [Tra50] however establishes the undecidability of first-order logic on finite models. And even for conjunctive queries, the problem of (Boolean) query answering is NP complete [CM77], since generalising on satisfiabilty problems. Therefore, various restrictions of conjunctive queries were studied, for which (Boolean) query answering is in

P-time [Yan81]. In other cases where recursion is needed, even FO-queries are not enough, so that richer logics with fixed points are to be considered, most typically Datalog [GK04] or monadic second-order (MSO) logic [Cou09, EC12, Bag06]. This happens for instance when interested in accessibility queries in graphs.

First-order logic exploits **data values** by adding equality relations either between elements (like joins), or with a set of constants (like finding all elements with a specific value). However, this poses a real problem for static analysis' purposes, as satisfiability of first-order logic with a predicate for data value equality is undecidable, even on strings [BMS$^+$06].

The situation becomes easier for purely **navigational queries**, that ignore the data values of the relational structures. For instance, the XPath query formalism exists in three main versions: XPath 1.0, XPath 2.0 and XPath 3.0. If we restrict these to navigational aspects, we obtain nice fragments, called core XPath 1.0 (or navigational XPath) [Md05, BL05, Lib06, BK08], and core XPath 2.0 (or conditional XPath) [Mar05, tCM07, tCFL10], whose behaviours are now well understood. In particular, query answering algorithms for Core XPath 1.0 can be defined by compiling navigational queries into monadic Datalog [GKP05], which, in the case of trees, has the same expressiveness as MSO or tree automata.

Finite state machines such as **tree automata** have many nice properties that are highly relevant to symbolic machine learning based on methods from grammatical inference [GO93]. The two most important properties are that tree automata can always be determinised, and that minimal deterministic automata are unique (up to state renaming). This provides for any regular tree language a unique normal form, as described by the Myhill-Nerode Theorem [Myh57, Ner58]. Such unique normal forms are essential for identifying regular languages from positive and negative examples [GO93]. How this can be exploited for the learning of navigational database queries in trees and graphs is one of the two main topics of this thesis.

The other main aspect of this thesis concerns the inference of **tree transformations**. Again, first-order logic proves to be a nice tool to understand those processes, as interesting fragments of XQuery have proved to be equivalent to first-order definable transformations [BK09]. First order fails however to capture recursions in a proper way. This can be done by adding ad-hoc predicates that capture the closure of other relations, such as 'descendant' for the closure of the child relation [Pot94]. This remains however very limited.

Monadic Second Order logic - or MSO - becomes then a natural candidate as it can express monadic recursion. MSO also allows to obtain a nice formalisation of a large class of graph transformations [EC12]. MSO can in general express formulae for which even evaluation becomes PSPACE-complete. However, in some restricted situations this problem becomes tractable - for instance it becomes linear time for graphs of bounded

tree-width [Cou09]. This condition is respected in particular for data organised as trees.

On the side of finite state machines, tree transducers constitutes a family of formalisms that can perform tree transformations efficiently. This wide family includes **deterministic top-down tree transducers**[TW68, Eng75, CDG+02], which constitute maybe the most classical and standard class. It is worth noting that some results of tree automata translate to this class, notably the existence of a normal form [EMS09]. The work we present here starts with these results and uses them to adapt existing learning algorithms for word transducers [OGV93] to obtain learnability results for deterministic top-down tree transducers.

Our next goal is then to extend this result to obtain learnability results for larger classes of transformations. In particular the class of **Macro Tree Transducers** with **look-ahead** [EV85, Eng77] constitutes a target of choice, as it captures the class of MSO definable transformations [EM05]. To obtain a learning algorithm for this class, we first need to study learnability of both macro tree transducers, and of tree transducers with look-ahead. This is what constitutes the approach we develop here.

3. Related work: Machine Learning for databases

Machine learning provides a wide array of techniques that can be used to analyse and exploit data in databases. The techniques we develop are part of this family, with a novel approach. We present a quick tour of tools that machine learning can offer to database management, and explain how our techniques are integrated in this field.

Two main scenarios of learning are usually considered in the field of machine learning: unsupervised and supervised learning. First, in **unsupervised learning**, the inference system has access to the whole set of available data with no further directions on how to exploit them. The typical goal is to cluster the data into intelligible pieces, or to infer implicit rules on how the data is organised, such as finding patterns or statistical dependencies between several attributes. In the database world, this task is usually called **data mining**.

Data mining consists in performing an analysis of data that usually includes several steps, such as data management, preprocessing, data visualisation, ... But the core step is to identify patterns, regularities or, more generally, knowledge from data, and this is mainly based on machine learning techniques. For this, many possible learning techniques can be used, typically based on statistical approaches: **bayesian networks** [BG07], k-**nearest neighbours**, **maximum entropy** models [Bor99], **cluster** analysis [ELL09], ... Data mining can be seen as some kind of query inference, where nearly nothing is known on the query, except the data on which it operates on.

Statistical analysis can also benefit from extra information. In this situation, called **supervised learning**, the user has a better knowledge of the query he wants, and the inference process is enriched either by information on the queries (such as a sample of desired results) or by interaction with the user to define the query. The process can also sometimes analyse the whole database and associate both supervised and non-supervised learning, in what is usually called semi-supervised learning [CSZ10].

In supervised learning, many statistical techniques can be seen as generalisations of the classical linear regression algorithm. The general idea is to project the data into a space where the query is seen as a function that can be extrapolated, using techniques such as **neural networks** [Bis95], **conditional random fields** [LMP01], **Decision Trees** [Qui87], **Maximum Entropy**, **Hidden Markov Models** (or HMMs) or **Support Vector Machines** (or SVMs [SS01]). In SVM for instance, the different attributes of the data are combined together into a space with basically as many dimensions as possible such that data become separable by a hyperplane. The inference of this hyperplane is made possible in practice by the fact that a technique - called the kernel trick - avoids to actually compute the data projection.

In information extractions, all those approaches have been tested with some success on the problem of query inference, in particular decision trees [Mar07], hidden markov models [FM99], Maximum entropy [CN02] or Support Vector Machines [IK02, MMP03].

However, all those approaches typically have difficulties to operate using structured or semi-structured data, and to infer concepts as complex as regular queries as the search space is too vast. Some techniques attempt to tackle the problem directly and extend techniques, such as SVMStruct [THJA04, JFY09], but still face important complexity issues which render the problem intractable on real datasets. Another approach is to reduce the input document to a finite (but possibly big) set of features, as in [FK00a, GMTT06]. This approach allows to encode a wide variety of information about the data, however, it does not allow to capture classes of concepts as complex as those captured by finite state machines.

On the other hand, **symbolic machine learning** techniques aim to infer directly complex classes of concepts, such as one represented by grammars, complex logic formulae, or finite state machines. They suffer some drawbacks with respect to statistical approaches, such as being less robust to noisy data, which make them harder to use in many practical cases. However, we will see that the framework we use to infer queries will avoid some of those difficulties, and hence, make this family of techniques particularly attractive to query inference.

In information extraction, first attempts of establishing symbolic machine learning algorithms ignored the tree structure and considered the document as a simple text, defining the query - called a wrapper - essentially as

a **regular expression**, or a set of regular expressions, that define the part of the text to extract. See for instance [Kus97, MMK99, Sod99a, CRdR00] Those techniques basically extend methods used for information extraction from text. The fact that they ignore the arborescent structures however greatly weakens them, and modern approaches use more heavily the tree structure of the documents.

Inductive Logic programming (or ILP) techniques have also been tested on information extraction with systems such as SRV [Fre00], Rapier [Cal98] or WHISK [Sod99b] which directly infer a system of logical rules that mix a textual analysis and a local view of the data structure. Those systems are also sometimes used as an assistance to statistical approaches, as in [RJBS08] or in BWI [FK00b]. Although these techniques have the advantage to exploit the textual content of the document, they fail to capture the class of queries we aim for as they have in fact a very limited use of the document structure.

Some work then began to exploit more fully the tree structure by defining queries using **tree patterns**, or extensions of these - such as k-testable languages which are more general tree languages defined as a succession of patterns [KVBB02, KBBdB06, RBVdB08]. Those have proved to obtain better results than the simpler tree patterns approach, but still capture only a subset of the regular tree languages.

The techniques I present in the following of this document will go one step further, as I will present methods that infer queries represented by tree automata, and hence capture a much richer class of queries than previous works presented above.

4. Contribution: Learning Queries and Transformations

The design of a query is a task that requires expertise. To be able to create a query, one must understand the data's structure and the query language. We already argued that it is harder for the user to understand precisely the data's structure in the field of semi-structured data than for other data models in particular because schemas are not always clearly defined. Query language are also a difficulty as many formalisms exist for various applications, and their expressiveness is typically greater than in relational databases for instance.

For those reasons, our contribution essentially aims at helping the user by replacing the task of query conception in a formal language with some simple interactions with a machine learning process that creates the query itself. As those interactions provide clean data (we assume the user does not make mistakes), this permits to use symbolic machine learning techniques that we develop and that allows the exact learning of the target query (as opposed to approximate learning which is more the norm in statistical learning).

This interactive setting is close to the learning setting defined by Angluin in [Ang87a, Ang88], a context in which concepts such as regular word languages has been proved learnable using some simple interactions with an end-user - or more generally to an oracle which can be of any nature. We essentially extend this approach to capture classes of queries. We first define algorithms that infer queries, and then extend it to the case of query-based transformations. We first focus on tree structured data, but we also propose directions to deal with graph structured data.

Learning Queries

As tree automata can represent any MSO formula on tree structured data, they can in particular be used to operate any MSO-definable query on trees. This is done by first representing the query as a tree language that contains trees annotated with the query answer. This allows to define MSO queries as regular languages. For this purpose, we use [CNT04a] Node Selecting Tree Transducers (NSTT), which are top-down tree automata that recognise tree languages of queries and which operate on a Currification of unranked trees. NSTT can also be used to annotate an input tree with respect to the query, which allows to obtain the answer of the query.

NSTT inherits the properties of tree automata. This includes a Myhill-Nerode construction that allows a minimal normal form [Myh57, Ner58]. This is the basis for learning algorithms such as *RPNI* [OG92, OG93] or L^* [Ang87a]. *RPNI* allows for the inference of a regular language of words [OG93] or of trees [OG93] from examples and counter-examples, while L^* infers a regular language of words from interaction with a user.

To infer queries, we needed to extend those results to queries represented by NSTT. A first theoretical result on the inference of monadic queries in a non interactive setting (inference from a sample) has been first presented in [CLN04]. This work has then been extended to an interactive setting (inference by interaction) in [CGLN07] where the user gives only partial annotation. For instance, if one wants to define a query that extracts product names on an e-commerce web page, the user starts by selecting only a few of them, interacts with a machine learning algorithm that defines a query which is then tested again on the document. The answer of the inferred query may then be corrected by the user (that provides a few more annotation for instance), and the process can be iterated until the inferred query is considered correct.

Those works have been then extended in [CGLN08b] where we improved our approach. In particular, in some situations a schema is available, typically a DTD. We investigated how this knowledge can help the learning. This uses an efficient algorithm for inclusion test between languages defined by tree automata we defined in [CGLN09]. Second, we defined formally how the learning algorithm can prune input trees to focus on parts for the queries, and how this affect the class of queries that is inferred in [CGLN10].

Learning Transformations

As transformations are based on formalisms close to node selection queries, the same kinds of difficulties arise. In particular, direct induction of logic formulae remains a difficulty. Using finite state machines seems a more promising lead, and that is the way we engage. In particular, we can rely on techniques that proved deterministic word transducers to be learnable [Cho03, OGV93].

In the large family of tree transducers, we focus our study on top-down tree transducers, which seems to be the corner stone for further work. In particular, macro tree transducers (MTT) are tree transducers that are extended with a mechanism of memory that can store partial outputs. As argued before, MTT, when added with regular look-ahead (a tree automaton makes a first run on the trees to annotate them) is proved in [EM03] to be more expressive than MSO definable transformations. In fact, the two classes are equivalent when MTT with regular look-ahead is restricted in the way they use copies in the output. This makes macro tree transducers a valuable target for our study.

Our first important result, presented in [LMN10a] is the elaboration of an efficient learning algorithm for deterministic top-down tree transducers, a result based on a Myhill-Nerode theorem that we also defined. Extensions from this class to richer classes, in particular macro tree transducers with regular look-ahead, is a future work but two important intermediary results will be presented. First, the use of look-ahead in inference is investigated in [BLN12], in which we present an efficient learning algorithm for word rational functions, represented by word transducers with a regular look-ahead. Second, we investigate in [LLN$^+$14] the inference of sequential tree-to-word transducers, which are transducers able to deal with concatenation in the output, a mechanism which is central for MTT.

Learning for Graph Structures

After trees, graphs constitute the formal framework for several semi-structured database formats. In particular, the RDF format is certainly the most iconic graph format, used primarily in a semantic web context. The main querying language for RDF is SPARQL, a language that allows not only basic Boolean operations, but also recursions.

After query inference on tree structures, our next goal is to extend our work on to query inference on graph structures. One of the most basic case is to use **Regular Path Query** (RPQ) to define monadic queries [CMW87b, CGLV02]. These are regular languages of words that select nodes of the graph from which one can draw a path that describes a word of the language.

We already approached the problem of RPQ learning in [BCL15a, BCL15b], in which we studied both a static and an interactive setting. This allows to obtain first results, but also highlighted some difficulties intrinsic to graphs. In particular, when a node is given as an example of the query, it is not trivial to know which one of its outgoing paths is responsible for its selection. To solve this issue, we had to either apply restrictions to the class of RPQ (by limiting lengths of paths for instance), or use more informative interactions with the user than the one we developed for tree queries.

Our results on RPQ are based on monadic queries, but they can be easily extended to binary queries as well. The extension from the monadic case to the n-ary case is also possible as they can be considered as combinations of binary queries.

RPQ constitutes a first step for graph query inference. More complex cases would be to use regular languages of trees to define queries, or potentially even more complex structures, that we also want to investigate in the future. For instance, we believe that an MSO query that selects a node in the graph can be expressed as some kind of regular tree language that would select a node if its - potentially infinite - unfolding tree is in the language. If this is correct, and providing we manage to also operate on a finite portion of this unfolding - we think that this could allow to adapt our technique for query inference on tree structures.

Another limitation is the fact that we do not exploit data values of the structure. This limitation already occurs in the tree case, and is certainly an important one as data values are usually an important element used in queries. However, in a node, it is usually not the data in itself that is important, but the relation it has with other nodes, or with a value present in the query. In essence, this is similar than dealing with equivalence relations, although it sometimes corresponds to other comparison relations.

We see two ways of dealing with this. The first one is simply to consider this relation as a new edge on the graph. This allows to exploit previous results with no changes. This technique can of course only be used for graphs and not directly for structures such as trees. Another solution is given by [KF94a] with register automata - also called data automata. In the case of words, these automata read sequences of values from an infinite set (sequences of integers for instance). The automaton is then able to store some values, and later compare it with another value. For example, it can verify that a sequence starts and ends with the same value. This class has many good properties, such as a normal form, a Myhill-Nerode theorem [MMP13], and even some learnability results [BHLM13, DHLT14]. We plain to extend those results to transductions and to tree structures in order to have learnability results for classes of tree or graph queries that can exploit data values.

Finally, we want to aim at **graph transformations**. This problem, which is central in data-exchange [DHI12] for instance, consists in adapting

data present in a graph to use them within another structure. In data exchange, edges of the new structures are defined by logical formulae on the input structures. Much in this spirit, we want to define edges on the new structures as binary queries on the input query. In this sense, this can be considered as a direct adaptation of our work on tuple queries. However, the learning context is different, and a new learning scenario has to be invented.

5. Research Organisation

Works presented here have been initiated within the **Grappa** Team, a research team of the LIFL (Laboratoire d'Informatique Fondamentale de Lille, now CRIStAL) and of the University of Lille 3 (now part of University of Lille) focused on Machine Learning. From 2004 to 2012, it has been part of the **Mostrare** team-project of INRIA and LIFL, whose main topic is machine learning in structured documents (mainly tree structures). From 2013, I integrated into the **Links** Team project of INRIA and CRIStAL, which is focused on studying linked data.

These works have been developed partially through several PhD thesis that I co-supervised. It has also been part of projects of the *Agence Nationale pour la Recherche* (ANR). We make a quick resume of both here.

PhD Thesis

Works presented here have been developed during six PhD theses, that we present in a few words.

Julien Carme [Car05], defended in (2005). During this thesis, we first investigated inference of monadic queries represented by tree automata from completely annotated documents in [CLN04]. A first definition of queries as regular tree languages has been proposed, based on Node Selecting Tree transducers (NSTTs), a transducer formalism that can either be seen as deterministic tree automaton that recognises the tree language of the query, or as a querying mechanism that extracts nodes of a tree according to the query. Also, queries represented by NSTT have been proved to be learnable from complete annotation, i.e. when the learning algorithm operates from a sample of trees for which all answers of the target queries are known.

This work has then been extended to an interactive setting in [CGLN05], where the query is inferred from simple interaction with a user: the user provide some annotation from a document. This approach has been validated by a prototype and experimentation that indicates that inference of queries can be done in an efficient way, and with very few interactions.

Jérôme Champavère [Cha10], defended in 2010. Work of this thesis essentially extends in the same line, focusing mainly on the usage of schema in inference. This is investigated notably in [CGLN08b], which indicate that using this information greatly helps learning. This is particularly relevant when one wants to infer queries for XML, where the schema is usually known as a DTD (as opposed for instance in inference of HTML queries).

This has been made possible thanks to a new algorithm to check the inclusion of tree automata published in [CGLN08a] that allows to test whether a language represented by a tree automaton A (not necessarily deterministic) is contained into a language represented by a deterministic tree automaton (bottom-up or top-down) can be done in $O(|A| \times |B|)$, previous result being in $O(|A| \times |B| \times |\Sigma^n|)$ (Σ being the alphabet, and n the maximum arity). This result is extended to inclusion for unranked tree languages and inclusion within an XML DTD, which is a corner-stone in our learning algorithm.

Grégoire Laurence [Lau14], defended in 2014. This work investigates learning of sequential tree-to-word transducers, a natural model of transducers that inputs trees from a regular tree language and outputs words from a context free grammar. As trees can be serialised, this model is more general than tree to tree transducers. Also, it implements concatenation in the output, which makes it an interesting object of study.

The first result of this PhD Thesis is to study the equivalence problem for this class in [SLLN09]. It has first been done by establishing that equivalence of deterministic tree-to-word transducers, deterministic nested-word to word transducers (also called visibly push-down transducers [RS08]), and morphism on Context Free Grammars are all polynomially reducible problems. Note that the latter problem has been proved to be polynomial in [KP94].

This study has been pursued in [LLN+14], where we defined an efficient learning algorithm for this class: the algorithm takes pairs of input tree / output words and aims to output a deterministic tree-to-word transducer in a normal form.

Radu Ciucianu [Ciu15], defended in 2015. This work studies interactive learning of queries for databases, with two important aspects: learning SQL queries, and learning queries for graphs (mainly RDF). My participation is on the second point, where we study learning scenarios for the class of Regular Path Queries (RPQs), which are queries where a node is selected if there is exists a path that starts from it which is part of a specific regular expression. This serves as a basis for more complex queries in RDF. Both a static and an interactive setting have been developed in [BCL15b, BCL15a], where a query can be inferred either from a set of nodes of the query or by asking the user to correct the answers of an interred query.

Antoine Mbaye Ndione [Ndi15], defended in 2015. This work involves property testing in semi-structured testing. The main results allowed to define the first efficient algorithm to estimate with good confidence if a (possibly very big) XML document satisfies a DTD in sub-linear time, i.e., in a time that only depends on the depth of the tree and is polynomial in the size of the DTD [NLN13]. It is based on a mechanism of random sampling of the tree that builds an approximate representation of it on which membership can be tested.

While different of other works presented here in the sense that it has no direct link with learning or queries, this work allows to give some insight on learning tree languages, and in particular tree queries, in a more statistical approach.

Adrien Boiret [Boi16], defended in 2016. In this thesis, several topics related to tree transducers are explored. In particular, we investigated the inference of look-ahead mechanism. This allowed to obtain learnability results in the word case for rational functions in [BLN12].

Works of Adrien Boiret also allowed to have a better understanding on equivalence and learning of Deterministic Top-Down Transducers, and studied in particular the role of the domain in those mechanisms. They also made significant improvements in normal forms and learning of tree-to-word transducers.

ANR Projects

These works are also integrated into several ANR projects, that we quickly detail here, as well as highlighting some activities in them directly in link with works presented here.

First, the **Marmota** project (2006-2008) dealt with stochastic tree models and transformations. It was a collaboration between LIP6 (P. Gallinari), LIF (F. Denis) and the university of St Etienne (M. Sebban). The project aimed to study problems raised by XML data manipulation with three main domains: formal tree languages, machine learning and probabilistic models. Works inside the Marmota project included learning of queries with schemas and n-ary queries.

The **Codex** Project (2009-2011) aimed at studying Efficiency, Dynamicity and Composition for XML Models, Algorithms and Systems. It was a collaboration with team GEMO (INRIA Saclay), team WAM (INRIA Grenoble), LRI (University of Orsay), PPS (university of Paris 7) and university of Blois.

The **Lampada** project (2010-2013) aimed at studying machine learning on structured data. It focused on high dimension learning problem, large sets of data, and dynamicity of data. It was a collaboration between team

Mostrare and Sequel of INRIA Lille, LIF of Marseille, Hubert Curien Lab of Saint Etienne, and LIP 6 of Paris.

The **Colis** Project (2015-2019) aims at applying techniques from logic and tree transducers to analyse shell scripts, and in particular installation script of UNIX systems. It is a partnership between INRIA Lille (Links team), INRIA Saclay (Toccata team) and university Paris Diderot (IRIF laboratory). Works in this project include defining models of tree transducers that can represent classical operators on file systems (copy, move, rename, ...) and for which statical analysis can be done efficiently (in particular composition of transformation, equivalence, emptiness). Typical problems include building a tree transducer that represents a whole script (built by composing transducers that encode elementary operations), or for instance, compose an install script with an uninstall script and check whether the result is truly equivalent to identity.

2 Tree Queries

The first aspect of our project starts with two observations. First, regular queries can be represented by tree automata, and second, grammatical inference techniques offer a nice framework for tree automata inference. We combine both points to define an algorithm for query induction.

The starting point of our approach is to represent queries by tree automata. This first originates from [NPTT05] which provides a representation of queries as tree languages. From this, we defined the model of **Node Selecting Tree Transducers** (or NSTT) in [CGLN07], a formalism that can be seen either as a tree transducer that performs the query or as a tree automaton that recognises the representation of the query as a tree language. We present this model in the first section of this chapter.

Particularly interesting for this work, this initial approach allowed to define first instances of learning algorithms for tree queries. However, first attempts to use those algorithms on real data proved to be somehow disappointing. The reason turned out to be that NSTT are automata that encode whole input trees, including parts irrelevant to the query. This makes their learning harder than necessary. To avoid this problem, we developed **pruning techniques** that remove unnecessary parts of input trees, and allowed a practical use of our techniques. The resulting formalism, pruning NSTT, are presented in Section 2.

In the third section of this chapter, we present the **learning algorithms** that we defined. We based our approach around the RPNI-Lang Algorithm [Lan92, OG92], a learning algorithm initially used to infer regular languages of words, but that have been extended to infer regular languages of trees in [OG93, BH99]. We start by presenting how the *RPNI*-Lang algorithm could also be adapted to infer NSTT from documents completely annotated by the user. As a second step, we indicate how pruning techniques can be integrated in the learning process. This allows to perform the inference from documents which are only partially annotated, which considerably reduce the amount of data needed from the user.

The last step of our approach consists in integrating our learning algorithm in a convincing scenario. In the last section of this chapter, we describe a process that uses **simple interactions** with the end-user to build the sample needed by the learning algorithm.

1. Regular Tree queries

This work starts with the observation that tree queries can be represented by tree languages. This allows to use tree automata to represent queries. We will call regular queries the class of queries that can be defined this way, and we will see that this class corresponds to the class of tree queries definable by MSO formula.

After specifying the tree model that we need, we indicate how to represent tree queries by tree languages. The tree languages will be composed by trees annotated to indicate which nodes are selected by the query. For monadic queries, one can simply label every node by a Boolean that indicates whether the node is selected or not. For an n-ary query, every tuple answered by the query is encoded by a specific tree.

We detail here those two encodings, and present here tree automata that can recognise those encodings, and hence, represent queries. Note that those elements, crucial for the following, have been presented first in [CNT04b] and are not part of our contributions.

Our Tree Model

Trees are essentially directed graphs with a specific structure: they have a root, and every node but the root has exactly one parent. They are particularly adapted for storing documents, and they also allow for faster access than general graphs. These are two of the main reasons why they are at the core of several semi-structured formats, such as XML or JSON.

There are several possible models for trees. We choose to use one that is adapted to finite state machines, as that will be our favoured tool. As a consequence, we make the following choices:

- First, each tree is considered **ordered** - children of a node follow a specific order. In semi-structured data, this is usually not a restriction as there is often a natural 'document'-order, even though this order is not always meaningful.
- We use a **finite alphabet**. For semi-structured data, this is not the case in general: leaves contain unrestricted data, and even internal symbols are in the most general case not limited to a finite alphabet. However, in many practical situations, the number of possible internal symbol is indeed restricted to a given alphabet - typically defined in a schema. For leaves, as argued before, queries do not usually deal with the data themselves but rather with the relations they have to each other (such as joins) or with external constants (like looking for a specific value given by the query). As a first step however, we ignore this aspect and simply use a generic symbol for leaves. This turns out to be sufficient in many applications, but we will see in the end chapter suggestions on how to avoid this restriction.
- we use **ranked trees**, this means that each symbol used to label a node always has a fixed number of children. In fact, we even mainly use binary trees. As it is well known, this is usually not a restriction, as trees that do not follow this rule (as typical XML or JSON trees) can be encoded into binary trees. The most typical way to do this is

to use the 'first-child / next-sibling' encoding. However, this encoding is not adapted for our purpose as a bottom-up reading of such an encoded tree does not correspond to the natural processing of a tree in our applications. We use instead the **Curryfied encoding**, that sees a tree as a recursive function call, and transforms each symbol as its binary Curryfication, using a special 'Curry' operator well known in functional programming, and that we denote @. For instance, $f(a, b, c)$ becomes $f@a@b@c$, or, if viewed as a binary tree, $@(@(@(f, a), b), c)$. This encoding is more adapted to bottom-up operations as a bottom-up traversal of such an encoded tree allows to see all internal labels as leaves.

Queries as Tree Languages

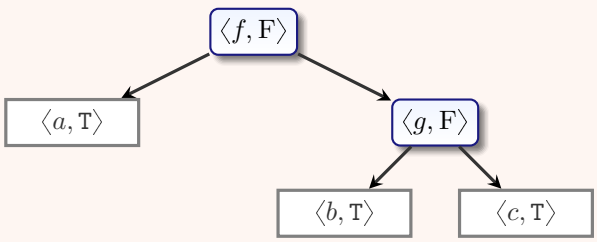
Based on our notion of trees, we define n-ary tree queries first as functions that take as an input a tree and output a set of n-tuples of its nodes. One of the first ideas that we use, and that originates in [NPTT05], is tuple queries can also be defined as tree languages.

First, for **monadic queries**, one simple thing can be done. As a monadic query is a function that takes as an input a tree and outputs a set of its nodes, one can without loss of generality consider the query as a function from an input tree to another tree with the same set of nodes but decorated by Booleans that contain either T or F depending whether the node belongs or not to the answer of the query.

This way to present monadic queries allows us to easily transform the query into a tree language: it will contain input trees that have for each node an additional Boolean annotation indicating whether the node is selected or not. We call this representation the **Boolean representation** of the query.

Example 1.

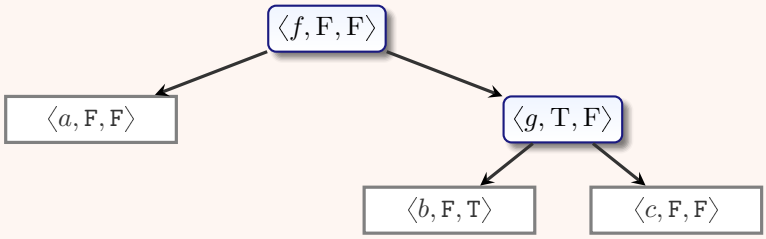
Consider the query that selects all the leaves of a tree. In the tree $f(a, g(b, c))$, only nodes labelled a, b and c will be annotated positively, which would give the following tree. Note that in this representation, any unannotated input tree has actually exactly one corresponding annotated representative in the query language, the one whose annotations correspond to the answer of the query.



This technique can not be applied in general for n-ary queries for n larger than 1, but we can adapt it. Indeed, n-ary queries in general associate a set of node tuples to each tree. Each of this tuple can be translated into an annotated tree that encodes it. Those annotated trees are obtained by taking an input tree, and adding to each node a vector of n Booleans as an additional annotation. This vector of Boolean indicates for each node if it is selected in the output tuple, and if so, in which position - for instance, a node annotated by the vector $< \mathrm{F}, \mathrm{T} >$ is actually the second node of the output tuple. That means of course that there are no two nodes that have the same position of their Boolean vector annotated by T - but note that a node can be selected in several positions of the tuple. We call this representation the **canonical representation** of the query.

Example 2.

Consider for instance the query that outputs pairs of nodes which are in relation parent - children. On the input tree, the pair consisting of the g node and the b node would be one of the four output tuples. The canonical representation of this tuple will be the following tree:



Note that the tree language that represents the query contains actually four trees corresponding to the input tree $f(a, g(b, c))$, one for each output tuple.

As a consequence, the canonical representation of a query will contain, for each input tree, exactly as many annotated canonical trees as there are answers of the query for this tree.

Note that monadic queries naturally accept both Boolean and canonical representation. Usually, Boolean representation will be preferred as functionality of the query will be easier to check (for each input tree, there is at most one output annotated tree).

.

Node Selecting Tree Transducers

Contribution

This section presents **Node Selecting Tree Transducer**, our model for representing tuple tree queries, which we introduced first in [CLN04] for the monadic case and in [LNG06] for the n-ary case.

As queries can now be represented as tree languages, it is natural to call regular a query whose tree language (Boolean or canonical) is regular, i.e. recognisable by a bottom-up tree automata. Note that for monadic queries, Boolean and canonical representations both lead to the same class of regular queries [NPTT05].

Regular queries correspond to query languages definable in Monadic Second Order Logic [TW68], a class whose expressiveness seems adequate for practical queries, as argued in our introduction. They also inherit many good properties from tree automata. First, they have a canonical representation, which is the minimal bottom-up tree automata that recognises them. As such, equivalence can be checked efficiently for instance. Also, given a tuple of nodes, it can be checked in linear time whether it is part of the query answer or not (just check if the corresponding Boolean or canonical tree is in the language).

Ultimately, we want to use trees that comes from XML or JSON documents for instance, which are unranked. So, our tree automata need to deal with trees that are in fact encodings of unranked tree. As our approach will be bottom-up (because the notion of determinism is better in term of expressiveness than top-down approaches), we focus on Currified encoding. Tree automata that operate on Currified encoding of unranked trees are introduced in [CNT04a] where they are called **stepwise tree automata**.

Deterministic Stepwise tree automata have several nice properties. In particular, they have a unique minimal canonical form, as deterministic bottom-up tree automata. Note that this is not the case for all finite state machine formalisms that deal with unranked trees, such as hedge automata for instance [Mur00, BKMW01].

We call the resulting model, i.e. deterministic bottom-up tree automata that recognizes Currified encoded unranked tree languages representing queries (canonical or Boolean), **Node Selecting Tree Transducers** - or NSTTs. The reason behind this name is that they accept two dual representations. Indeed, they can be used as a tree automaton that recognises the tree language of the query. But they can also be used as tree transducers that computes outputs of the query, i.e. that selects nodes. They take a tree as an input and outputs annotated trees corresponding to the answer of

the query. To obtain this, one can simply consider the NSTT that ignores the annotations, using a simple projection. This - now non-deterministic - tree automaton recognises unannotated trees, and it has the property that each of its runs corresponds to a tree annotated by the query. Enumerating runs of this automaton on an input tree therefore provides all the output tuples of the query. The run of a non-deterministic tree automaton being still polynomial, NSTTs turn out to be an efficient representation for regular tree queries.

Queries as Pruned Tree Languages

Contribution

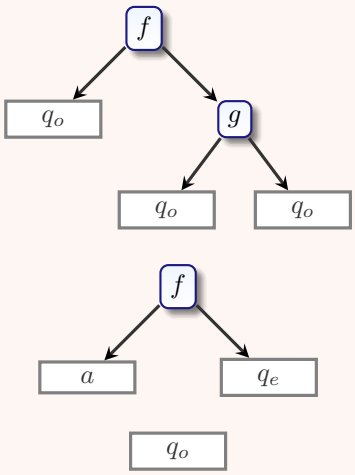
Pruning techniques presented here have been first published in [CGLN07]. These techniques were then improved in [CGLN08b] and in [NCGL13], in particular with the use of the schema in pruning.

Describing a query by a tree language as described above has the drawback that the language describes whole trees, even if large parts of those trees are irrelevant for the query. This makes the automaton that represents the tree languages potentially much bigger than necessary. This is a problem for machine learning as a bigger target automaton is harder to infer. To avoid this problem, we remove irrelevant parts of the trees using an operation called **pruning**.

Pruning a tree consists in removing whole subtrees of it and replacing them by a symbol. In the simplest case, the symbol is a generic special leaf symbol (we use $\perp$). Otherwise, one can use a symbol from a finite set that somehow represents or carries some information about the removed subtree. One solution for this is to use a deterministic bottom-up automaton D: a removed subtree is then replaced by the unique state that results from its evaluation by D. We will talk of **D-pruning**. The simple case (one generic pruning symbol) is then of course a special case where D is the universal automaton with a unique state $\perp$.

Example 3.

Consider a tree automaton D which checks the parity of the height of a subtree - which is the length of its greatest path to the root. A subtree of even height -such as $f(a, b)$ - would be put in state q_e while a subtree of odd height odd - such as $f(a, g(b, c))$ - would be put in state q_o. For the input tree $f(a, g(b, c))$, the three following trees are all valid D-prunings.



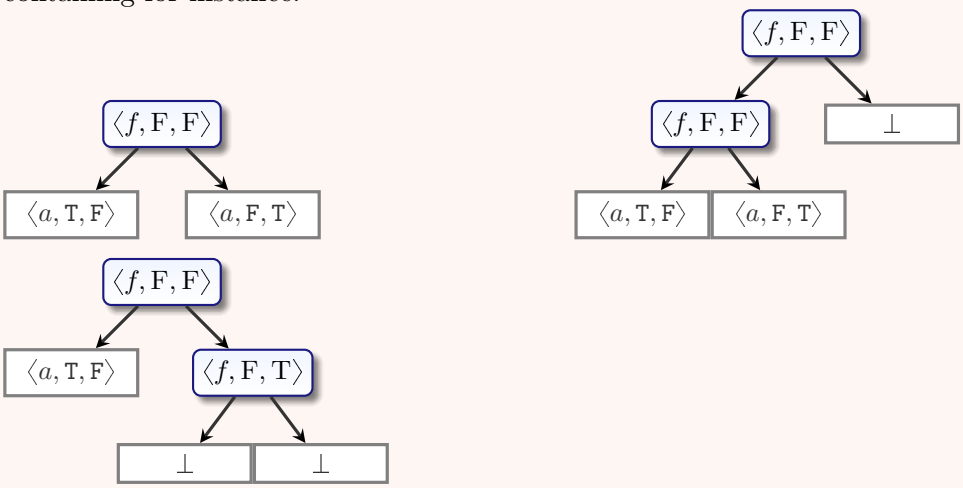
In general, any kind of deterministic bottom-up tree automaton (or DBUTA) D could be used for pruning, and there are certainly analogies between this pruning mechanism and what we will see later on with look-ahead of transducers. One convenient solution is to use an automaton that describes the domain of the query. This has the advantage that this domain automaton is usually known - or can be obtained from inference - and it naturally provides a classification of subtrees that proves to be useful in practice.

A tree can accept many different D-prunings, as the pruning automaton is simply there to indicate what to put instead of a pruned subtree, but does not indicate which subtrees to prune. Also note that pruning can be performed on trees even if they are annotated, in which case, the annotation is simply ignored by the pruning process. In particular, the automaton D operates on the unannotated projection of the tree.

A language L of annotated D-pruned trees can encode a query in the following way. For an input tree t, every D-pruning t' of t that matches an annotated D-pruned tree t'' of L provides, either an output tuple if L is canonically annotated, or a set of nodes if L uses a Boolean annotation.

Example 4.

Consider the trivial pruning that replaces a subtree by the symbol $\perp$. The query that selects all pairs of nodes which are the first and second child of an f-labelled node can be represented by a language of D-pruned trees containing for instance:



For an input tree, say $t = f(f(a, a), a)$, knowing the pruned tree representation of the query is sufficient to answer it. For instance, the tree t matches with the upper-right tree presented here (i.e., its projection on the input alphabet is a valid pruning of t), which allows to know that the tuple consisting of the first two 'a' of t is an answer of the query.

NSTTs can be extended to recognise annotated D-pruned trees in a rather direct way: we consider NSTT that directly operate on D-pruned trees, i.e. that recognise tree languages on $T_{\Sigma(D)\times\mathbb{B}^n}$ either canonical or Boolean. We call those NSTTs **D-pruning nstts**.

Using D-pruning NSTT to answer queries for a tree $t \in \mathcal{T}_\Sigma$ can be done in a way similar to NSTT. A first run is done by the automaton D that associates the pruning symbols (i.e., states of D) to every nodes of t. Then, a run of the D-pruning NSTT can be done in a non-deterministic way, reading either symbols of the tree or corresponding states of D. The complexity of the process is similar to a run of an NSTT, i.e., polynomial within the size of the input tree.

The class of queries defined by D-pruning NSTT in general is still the class of regular queries defined as before: pruning does not increase the expressiveness of the class. However, some queries can be represented in a more compact way with D-pruning NSTTs, as they do not need to encode the whole domain. Consider for instance the query that just selects the root node of every tree, it can be done with a very simple D-pruned NSTT that just prunes everything but the root, while a 'classical' NSTT would need to encode the whole domain of the query.

2. Learning Queries

We presented in previous sections NSTTs and pruning NSTTs which will be the target representation we use for query inference. We now present here query inference algorithms based on these representations. Our final goal is to define an algorithm that will infer a query from a few simple interactions with an end user. Essentially, we will ask the user either to give some annotations on a document, to correct some annotations provided by the learning algorithm, or to indicate that a document is correctly annotated.

The first step consists in inferring the query from examples of **completely annotated** trees, i.e. from a set of input trees for which we have the whole set of answers given by the target query. Inspired by the *RPNI* algorithm of [OG92, OG93], we first developed an algorithm for this setting in [CLN04] for the monadic case. This algorithm has then been extended to the n-ary case in [LNG06]. We start by a presentation of those results.

As we do not really want the user to provide complete annotations of the input trees, our second setting consists in inferring the query from examples that has been only **partially annotated**. For this, we have developed learning algorithms that make heavy usage of pruning techniques presented before. We introduced a first algorithm in [CGLN07], that has been extended in [CGLN08b] and [NCGL13].

Finally, we finish this section by a presentation of our **interactive setting** that we introduced in [CGLN05]. We will see that through some basic interactions, we can obtain the partially annotated examples we require. As we assume that interacting with an end-user allows to provide only correct annotations, this allows to operate in a framework that is suitable for our learning method while still being convenient to use.

Learning from Complete Annotations

Contribution

The learning algorithms presented here have been first published in [CLN04] for the monadic case and in [LNG06] for the n-ary case.

The first setting we investigate is the inference of queries from **completely annotated examples**. Our approach is based on a generalisation of the *RPNI* algorithm, a learning algorithm for regular word languages discovered simultaneously by Oncina and Garcia in [OG92] and by Lang in

[Lan92]. This algorithm has then been extended to infer regular tree languages in [OG93] in the following sense. It takes as an input a sample of examples of a target regular tree language as well as a set of counter-examples of it and always provides a deterministic tree automaton (bottom-up) at least consistent with it. It has the property that, when the sample is large and representative enough, its output is guaranteed to be the minimal deterministic tree automaton that recognises the target language. *RPNI* is also an efficient algorithm in the sense close to the model of learning with polynomial time and data [Gol78, dlH97]: first it operates in polynomial time, and second, for any regular tree language, there exists a characteristic sample - i.e. representative enough for the algorithm to succeed - whose cardinality is polynomial in the size of any deterministic tree automaton that encodes it.

As queries can be represented by regular tree languages, and NSTT are essentially deterministic bottom-up tree automata, *RPNI* can be used to infer NSTT. This is exactly what we do. However, one major difference arises in the way we deal with **counter-examples**. In this first approach, we consider that for an input tree, all annotations are obtained, i.e., we know all the tuples provided by the target query on this tree. In this case, all other tuples can be considered as implicit counter-examples. In *RPNI*, the algorithm tests at each step whether the current hypothesis is not in contradiction with a set of counter-examples, which would be in our case badly annotated trees. This means that the current hypothesis should not recognise one of the counter-examples. In our case, this means instead that the current hypothesis should not provide on one of the input trees an output tuple which has not been given explicitly as an example.

The learning algorithm also requires the domain of the query - given as a tree automaton. This is usually given - in form of a DTD for XML documents for instance - otherwise it can also be inferred by the *RPNI* algorithm for instance.

The resulting algorithm, called $RPNI_{nstt}$ and described in detail in [CLN04], has therefore the following key points:

- It takes as an input a **domain** D, and a set of **completely annotated trees**, i.e. pairs of a tree and a set of tuples of its nodes. From this, it first encodes them into binary trees (using Curryfied encoding), and builds a set of annotated trees, either canonical or Boolean.
- It is then based on the same basic mechanism as *RPNI*: it first builds an **initial tree automaton** that recognises exactly the set of annotated trees given in that input.
- As *RPNI*, it enumerates pairs of states of the tree automaton and tries to **merge** them. The merge is accepted only if the resulting automaton does not conflict with the input sample. In our algorithm, and unlike *RPNI*, this is the case if the tree automaton provides a new tuple for

an input tree, or if the tree automaton recognises a tree outside of the domain of the query.

- The algorithm ends when **all pairs of states** have been considered. In the end, we obtain an NSTT that corresponds to a query which is - at least - always consistent with its input. The algorithm runs in **polynomial time**.

We implemented and tested this algorithm in [CLN04] for the monadic case and in [LNG06] for the n-ary case. It also allowed us to obtain the following theorem:

Theorem 1.

The class of regular tree queries represented by NSTTs is identifiable in polynomial time and data by the algorithm $RPNI_{nstt}$, which means:

- The $RPNI_{nstt}$ algorithm takes as an input a sample of an annotated tree language and a tree automaton D and outputs an NSTT consistent with its input in polynomial time.
- For each regular tree query Q represented by an NSTT A, there exists a sample S_Q of cardinality polynomial in the size of A such that, from any finite sample S with $S_Q \subseteq S$ consistent with Q, $RPNI_{tree}(S, D)$ provides a tree automaton that encodes Q.

The setting described in the theorem is close to the classical setting of Gold learning within polynomial time and data [Gol78, dlH97]. There is one difference: in the classical setting, the size of the characteristic sample (here S_Q) is required to be polynomial. Here, we only require its cardinality to be polynomial. This is not something specific to our setting and already occurs for $RPNI$. This is due to the nature of trees: small tree automata can encode large trees. For instance, consider a tree language consisting of only one balanced binary tree of height n such as $f(f(a,a),f(a,a))$: this is recognised by a simple tree automaton of n states, but has a size of 2^n and learning this language would require at least this tree to be present in the sample.

Learning from Partial Annotations

Contribution

Works presented here have been first introduced in [CGLN05] and have been further improved in [CGLN08b, NCGL12, NCGL13].

Although it has allowed us to obtain a first theoretical result, there are two reasons why we want to avoid the above scenario. First, it requires a complete annotation of the documents, a task that could be time consuming for the user. Second, the target of the learning pruning is an NSTT that describes complete documents, whereas large parts of those documents are irrelevant with respect to the queries and whose learning may spoil the whole learning process.

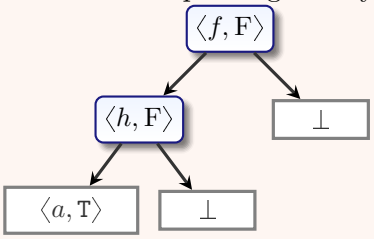
For those reasons, we prefer to infer queries from partially annotated trees, where the user provides only a subset of the answers of the target query, and prunes the rest of the document. Beside the obvious advantage that it reduces the amount of annotations the user has to provide on a document, this also reduces the size of the target NSTT, as it does not need to represent pruned elements of documents. And as the target of the inference is smaller, we will need fewer documents to infer it.

The input of our learning algorithm will now be **partially annotated trees**, with means that we now have pairs consisting of a tree and a set of tuples which are *some* of the possible answers of the target query on it. We start by encoding input trees into binary encoded trees, transforming them into annotated trees (canonical or Boolean), and then pruning them. This last step needs however to be specified. Indeed, we presented previously what a pruned tree is, but we have not indicated yet how to prune a tree.

There are indeed many possible ways to prune an annotated tree, that we call **pruning strategies**. These are formalised as functions that determine which environment of a node is relevant for the query. Technically, they are functions that take as an input a tree and a node, and provide a D-pruned tree that contains at least the node with its label, but all nodes which are assumed to be relevant for the target query. In experiments, we considered for instance the function $\texttt{path-only}_D$ (or simply `path-only` for a trivial D) which removes every node which is not on the path from the root to an annotated note, and the function $\texttt{path-ext}_D$ which keeps also nodes which are direct siblings of a node kept by $\texttt{path-only}_D$.

Example 5.

Consider the tree $t = f(g(a, b), h(c, d))$ with only the a-labelled node annotated T. The pruning of t by `path-only` gives:



The pruning strategy defines an environment around the annotated nodes, and aims at removing everything which is not relevant for the query. In fact, what this means depends of the query, and each pruning strategy actually corresponds to a class of queries. For instance, a query that would select every 'b' labelled node which is sibling of an 'a' labelled node can not be represented by a language of trees pruned according to $path-only$, as a crucial information would be erased. For a pruning strategy p, we call a query **p-stable** if it can be represented by a language of trees pruned according to p.

Pruning strategies define environments but can not be applied on annotated trees, as they can have several nodes positively annotated. For this, we need another notion that we call a **pruning function**. This is a function that takes as an input an annotated tree and prunes it. A pruning function is based on a pruning strategy in the sense that it should keep all nodes annotated by a T and their environment, as defined by the pruning strategy. The pruning function is typically defined as the union of all the node preserved by the pruning strategy, when applied on all T annotated nodes.

The learning algorithm itself - called $RPNI_{pruned}$- is an adaptation of the algorithm described in the unpruned case. It has three differences compared to it.

- First, as stated before, the sample it takes is only **partially annotated**: it contains pairs of a tree t and a set of tuples of its nodes containing only *some* of the answers of the target query on it.
- Second, it is parameterised by a **pruning function** p. The first step of the algorithm actually consists in applying this pruning function to its input. The algorithm will also test that the inferred language is indeed consistent with the pruning function, in the sense that all trees of the inferred language should indeed be trees that could be obtained by it. Note that we indicate in [NCGL13] that this test can be done in polynomial time, as it is essentially an inclusion test. Also, we focus on *regular* pruning functions, which can be seen as a special case of regular monadic queries.
- The third point is that, when using pruning functions, we do need **counter- examples**. These are pairs consisting of a tree t and tuples of its nodes which are *not* part of the answer of the target query. While on complete annotation, those counter-examples were deduced from the other annotations using the principle of functionality, this doesn't work anymore for partial annotations. The learning algorithm will therefore take as an additional input a sample S^- of counter-examples. The algorithm will have to test at each step that the target query answers in a manner consistent with the counter examples, i.e., that the inferred query does not output one of the counter-examples. This test is also polynomial.

This allows us to obtain the following result:

Theorem 2.

[CGLN05] For a deterministic tree automaton D and a regular pruning function p defined on D, the class of p-stable regular tree queries represented by pruned NSTTs is identifiable in polynomial time and data by the algorithm $RPNI_{pruned}$ in the following sense:

- The $RPNI_{pruned}$ algorithm outputs in polynomial time a pruned NSTT which is consistent with its input.
- For each p-stable regular tree query Q represented by a pruned NSTT A, there exists a sample S_Q of cardinality polynomial in the size of A such that, from any finite sample S with $S_Q \subseteq S \subseteq L_Q$, $RPNI_{pruned}(S)$ provides a pruned NSTT that recognises L.

Interactive Learning

Contribution

The interactive framework presented here have been published first in [CGLN05].

The scenario above explain how the learning process can operate from sets of partially annotated trees but do not describe how those trees are obtained. We use for that an interactive setting. Remember the basic idea of our approach: we want to help an end-user to define queries by replacing complex program writing with few simples interactions.

The most classical framework for interactive learning is the **Angluin learning model** [Ang87a]. In this setting, two kinds of interactions are used: **Membership Interactions** and **Equivalence Interactions**[1]. They were initially defined mainly in the context of language learning, such as regular word language. In this context, a **Membership Interaction** presents a word to the user and asks him whether or not the word is part of the target language. **Equivalence Interactions** present a representation of a whole language (in the form of an automaton typically) and ask the user if this corresponds to the target. The user then answers YES if it is or provides a

[1] Angluin uses the word *query* rather than *interaction*, we will avoid this terminology here as it has a different meaning for us.

counter-example (a word in the target language which is not in the inferred language for instance). In [Ang87a], it is stated for instance that regular languages are polynomially learnable in this setting, which means that an algorithm can infer the minimal automaton of a target regular language while using an amount of interactions which is a polynomial in the size of the target automaton, and with a polynomial time computation time.

Inspired by this model, we defined in [CGLN05] an interactive setting based on the following scenario: the user selects a document and annotates parts of it - he gives a few answers of the target query for this document. From this, the learning algorithm guesses a query and annotates the whole tree with it. The user may then correct some badly annotated positions if he is not happy with the answer, and iterates the process. This is done until the algorithm has inferred a query that behaves correctly on the tree, at which point the user may then test the query on the other tree - and eventually reiterate the process if the query behaves badly, or stop and accept the query.

Formally, this setting requires two kinds of interactions: **Correction Interactions**, and **Equivalence Interactions**.

The **Correction Interaction** is the main interaction: we present to the user an annotated document. The user then essentially answer **YES** if the annotation is correct with respect to the target query, if not, the user provides a set of tuples for which the annotation of the inferred query is not correct. One is sufficient, but we allow the user to provide several tuples during the same interaction.

The **Equivalence Interaction** is used to stop the learning process: we present a query to the user, and he answers **YES** if the query is correct, or he provides a document for which the query does not behave correctly. In practice, this query will be not be used directly, as we do not really want to present an NSTT to the end-user, as that would go against our policy of not requiring strong expertise (in particular not the ability to decipher a tree automaton). This is approximated by the fact that the user, after testing the inferred query on several documents and observed its good behavior, decides to stop the inference and keep the inferred query as it is.

The learning process goes as follows. It takes as a parameter a domain automaton D and a pruning strategy p. It starts with an empty hypothesis query Q_0 and a first tree (typically, a web page or an XML document). The user first provides some annotation on this tree - which can be considered as a first correction. This first annotated tree is given to the algorithm $RPNI_{pruned}$, which outputs a query Q'. This query becomes the new current hypothesis. The tree is then annotated by Q', and the result is presented to the user for correction. Either this is not correct, and the user provides some missing tuples or indicates that some tuples are wrongly annotated (which gives new counter-examples), and the process is iterated. If the annotation is correct, we perform an equivalence test, which consists in fact in either

stopping the process (if the user is happy with the current query), or by presenting a new document, annotating it with Q' and iterating the process.

This algorithm, called *SQUIRREL*, allows to obtain the following result:

Theorem 3.

[CGLN05] For a pruning strategy p, the class of p-stable regular queries is identifiable by *SQUIRREL* in polynomial time and by using a polynomial amount of Correction and Equivalence interactions.

Note that we also conducted in [CGLN05] several experiments that confirm the practical validity of our approach. We tested in particular the inference of classical queries on various websites, and have been able to infer them with only a few interactions (typically less than 10).

3. Conclusion

Works presented in this chapter have been the object of several publications that we summarise here. Most of them have already been mentioned above.

In [CLN04], we first established initial learning results for the class of monadic NSTT represented queries using completely annotated examples. A first implementation has been made, with a Mozilla plugin that allowed us to test the approach.

In [CGLN05, CGLN07], we introduced the notion of pruning NSTT, established learning results for this class, and specified the interactive learning setting based on correction and equivalence queries. Again, an implementation has been done, with subsequent experiments.

In [LNG06], we extended our approach defined initially for monadic queries to the n-ary case. In particular, this led to the definition of NSTT for n-ary queries, in particular with pruning. It also allowed us to define a learning algorithm for this class. An implementation and experiments have also been performed to validate the practical use of the approach.

In [CGLN08a], we defined an algorithm that tests the inclusion of a tree language over an alphabet Σ defined by a bottom-up tree automaton A into a language defined by a bottom-up deterministic tree automaton B. This algorithm allows us to obtain the expected complexity of $O(|A| \times |B|)$, where previous classical algorithm had a complexity of $O(|A| \times |\Sigma| \times |B|^n)$, n being the maximal arity of symbols in Σ. This result has also been extended to test inclusion of unranked tree language defined by a stepwise tree automaton A into the language defined by a DTD D with a complexity of $O(|A| \times |\Sigma| \times$

$|D|$). This result is the key to obtain the proper complexity in our learning algorithms.

In [CGLN08b], an extension of our pruning techniques have been elaborated to make use of the schema, a point that proved to improve efficiency of our learning algorithm, in particular in the case of XML documents with DTD. This approach have been improved in [NCGL13] with a better formalisation that includes the introduction of the notions of pruning strategies and pruning functions.

This work on query inference is the basic element on which following works are based. In next sections, we will see first how this contribution on tuple queries can be extended to infer transformations. We also started to extend works from tree queries to queries that operate on graph. Another point is that the most important limitation of what we presented here is probably that data values are not taken into account. We will also present works that intend to tackle this problem in later chapters.

3 Tree Transformations

In our introduction, we divided manipulations on tree-structured documents into two classes: queries and transformations. We present here how we extended results on query inference presented in the previous chapter to tree transformations, i.e. to functions that transforms an input tree into another tree.

We first present the class of transformations definable by MSO formula, which is the class that we ultimately want to infer. We then present our general methodology, and present learning algorithms for three classes of tree transformations. We finish by indicating how those three results all constitute one step towards the inference of MSO definable transformations.

1. MSO Definable Tree Transformations

Tree transformations are a classical problem of computer science. For instance, the compilation of a program is essentially a tree transformation process. In databases, this problem also appears when data is exchanged between several systems that uses different formats (different schemas). In the XML world, this is usually done with the XSLT or the XQuery language for example.

On the logic side, Courcelle gave in [Cou94] a characterisation of graph transformations based on **Monadic Second-Order** (MSO) logic. In this, MSO formula allow to define the edges of a new graph defined on a bounded amount of copies of the input graph (both nodes and edges). MSO Tree transformations are simply a restriction of this when both input and output are of tree forms.

MSO transformations have nice properties such as closure under composition, or preservation of context-free graph grammars. They also have good expressiveness as they capture whole FO but can also captures properties such as recursions that can not be achieved with FO formulae. As argued in the introduction, they offer an interesting formalisation of queries on semi-structured data and provide theoretical grounds for tree transformations.

On the operational side, finite state machines offer a variety of tools to perform tree transformations that complements logic. In this paradigm, a **transducer** is a machine that can essentially transform a structure into another. This originates from Mealy Machine [Mea55] or Moore machine [Moo56]. Essentially, these are automata that can produce a structure when reading the input, as it was defined in [TW68, Rou70]. Word transducers for instance are essentially word automata that associate an output subword to every transition, and sometimes also on final states: when the word transducer reads a word, it produces at the same time the sequence of all subwords produced by each transition, followed by the possible subword produced by the final state. Word transducers allow to define a class of transformations that is called **regular transductions**.

Tree transducers are essentially a generalisation of word transducers. The most classical models, **Top-down Transducers** and **Bottom-up Transducers**, are tree automata whose rules produce subtrees while reading each symbol of an input tree. In the non-deterministic case, these two formalisms are in fact similar, but things change when considering determinism. The classes of deterministic top-down transducers and deterministic bottom-up transducers both have interesting properties, such as a normal form (see [EMS09] for top-down and [FSM10] for bottom-up). However, the two classes of transformations that they define are unrelated.

Non-deterministic tree transducers (top-down or bottom-up) define an even larger class which can also be expressed by **deterministic tree transducers with look-ahead**: these are top-down tree transducers that are coupled with a deterministic bottom-up automaton - the look-ahead. The look-ahead makes a first pass on the input tree, annotating each node by one of its states. The top-down transducer then operates normally, using as an input for each node both its label and its associated look-ahead state. As both components of the transducer are deterministic, performing the transformation is an efficient process. However, unlike the word case, it is not known now whether this class has a normal form.

To further increase expressiveness, a possible solution is to add memory to states. A **macro tree transducer** (MTT) is a top-down tree transducer such that states are allowed to store output trees into some memory called parameters or macro which are also able to contain trees that are the result of the transformation of possibly another part of the input tree.

An important result, that comes from [EMS09], is that deterministic MTT with a regular look-ahead are strictly more powerful than MSO transformations. In fact, the two classes are equivalent when MTT are only allowed to perform a bounded amount of copies of the input.

MSO transformations seem to be an interesting target for our inference procedures, as they correspond to transformation classes of practical use and seem adapted to our learning techniques. Therefore, our long-term goal is to define normalisation and learning procedures for single use restricted MTT with regular look-ahead. That is the direction of the work we present here. It is worth noting that, up to our knowledge, no other work attempts to tackle the inference of this class with strong learnability results as presented here.

2. A Methodology for Learning Transformations

In the next sections, I will present several learning results for different classes of transformations. While they all present their own specificities, they also share some similarities which allow to use a somehow common methodology to obtain our learning results. We start by indicating our main guidelines.

The learning method used here is inspired by classical grammatical inference learning algorithms such as *RPNI* [OG92]. As we already saw, *RPNI* is an algorithm that infers regular languages represented by deterministic finite automata, and which is based on the Myhill-Nerode construction. It is in particular the algorithm that served as a basis for our inference algorithms of query languages presented in the previous chapter.

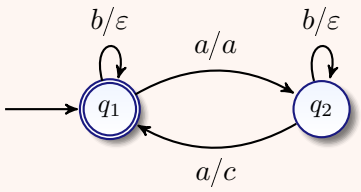
This approach has been extended to the case of word transducers with the *OSTIA* algorithm [OGV93]. This algorithm infers deterministic word transducers, also called **sequential word transducers** in [Sch77, Cho03], which are essentially deterministic word automata for which each transition is coupled with an output word. Each final state is also associated to an output subword. To obtain the image of an input word, one concatenates all the output subwords encountered during the run, stopping with the subword associated to the final state.

To obtain a learning result for this class, several steps have been necessary. We present them quickly as, even though this is not part of our work, all of the results we present in this chapter will be essentially generalisations of this. This will allow to present the methodology we follow, as well as presenting some notions we use.

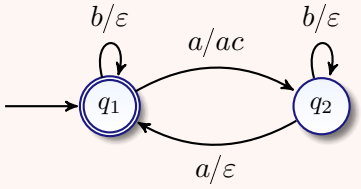
First, [Cho79] established a **normal form**, called onward or **earliest**, which is based on the idea that the transducer should produce its output *as soon as possible*. The idea to obtain it consists therefore mainly to *pull* all productions as much as possible. The earliest normal form can then be minimised by merging equivalent states to obtain a minimal normal form, that we call the **canonical earliest transducer**, which is unique for each transformation definable by a deterministic word transducer. This is illustrated in Example 6

Second, a notion of **residual transformation** has been defined. Consider a word $u.v$ and a transformation τ. the image of $u.v$ by τ ($\tau(u.v)$) can be decomposed in two parts: the partial image of u and the partial image of v. The residual transformation of τ by u, denoted $u^{-1}\tau$, is the function that transforms the word v into the partial image of v in the $\tau(u.v)$. Technically, it outputs the image of $u.v$ by τ and removes from it the largest common prefix of all images of words prefixed by u, which is assumed to be the partial image of u itself. This semantic definition of the residual transformation allows to define a **Myhill-Nerode like theorem**. This has two main impacts. First, for any transformation that can be performed by a deterministic transducer, the equivalence relation of the residual transformation has **finite index** (and vice-versa), which gives, as in the Myhill-Nerode theorem, a characterisation for this class of transducers. And second, one can build the **canonical earliest transducer** of the transformation from this finite set of residual transformations, associating one state to each distinct residual transformation. In this sense, residual transformations (or simply residuals) are semantic components of the transformation from which the transducer can be built.

Example 6.



A word transducer that takes as an input a string over $\{a, b\}$ with an even number of $'a'$ (the state q_1 is final with no production), outputs the string without the $'b'$ but replaces every second $'a'$ by a $'c'$ - transforming $'ababb'$ into $'ac'$ for instance. Transitions are labeled with an input letter and an output string - possibly empty, as denoted by ε.
This transducer is deterministic as its projection on the input alphabet is a deterministic automaton. It is however not earliest as the c could already be produced when reading the first a, as illustrated by this second deterministic transducer, which is in earliest normal form.



Finally, the **learning algorithm OSTIA** [OGV93, OV96] is defined from this Myhill-Nerode construction, simulating it from a finite sample. In particular, it associates to each transition the largest common prefix of all output words. The OSTIA algorithm allows to obtain learnability results, stating that the class of rational transductions is learnable when represented by earliest deterministic transducers in polynomial time and data. Formally, this means two things. First, OSTIA is an **efficient** algorithm, in the sense that it is a polynomial time algorithm: from any sample S, it outputs a deterministic earliest word transducer consistent with its input in $O(|S|^3)$. Second, it requires a **reasonable amount of examples** to infer, meaning formally that for each word transformation that can be represented by an earliest word transducer A, there exists sample S_A of size polynomial in the size of A - called characteristic sample for A - such that from any sample S of the transformation that contains S_A, $OSTIA(S)$ produces A - or a smaller transducer equivalent to A.

In order to reproduce similar results for other classes of transformations (with other classes of transducers), we follow essentially the same steps.

1. First, we need to define a **normal form** for the class of transducers. We will always base it around the idea of being **earliest**, although we will see that it can have different meanings depending of the class of transducers.

2. Second, we need to define a notion of **residual transformation** that allows to define a **Myhill-Nerode** theorem, in the sense that the distinct residual transformations should allow to obtain a constructive algorithm to obtain the normal forms. In our settings, those classes of equivalent residual transformations will correspond to states of the normal transducer. That also means that there should be finitely many distinct residual transformations, or at least, that a finite subset of those can be used in the construction of the normal form.

3. Finally, we want a **learning algorithm** that essentially tries to mimic the Myhill-Nerode construction, but using only a finite sample as an input. That also means that we have to define characteristic samples, i.e., samples that contain every information such that the learning algorithm is able to build the minimal normal form. Ideally, we also want characteristic samples to have a size polynomial in the size of the target transducer.

Beside this, we also need to take care of the **domain** of the transformation. Dealing with non total transformations has an impact on almost all levels, starting with the normal form. In particular, two residual transformations will be considered equivalent only if they are defined on the same domain. Typically, taking into account the domain causes difficulties which are essentially technical and that we will tend to ignore in this presentation for sake of clarity. However, we will detail this question when it appears that it has a significant impact on our formalisms.

We will apply this general methodology to infer three classes of transductions, that are all steps in the further goal of inference of MSO transformations. We will start with **deterministic top-down tree transducers**, then **transducers with look-ahead** and finally **tree-to-string transducers** later on. Although we will follow the above mentioned steps, we will see that each of those formalisms presents specificities that imply to alter our approach.

3. Contribution : Learning Top-Down Tree Transducers

Contribution

Results presented here have been first presented in [LMN10b]. These results have then been extended, in particular with respect to domain issues, which have been partially published in [BLN16] (learning within a regular domain). Other elements are under submission.

Our first goal is to establish learning results for the class of transformations that can be described by deterministic top-down tree transducers. This class of transducers, called here **top-down transductions**, will serve as a base case for our general approach.

Tree transducers, and in particular top-down tree transducers, constitute a long studied formal model - see for instance [CDG+02] for a more complete presentation - but it is [EMS09] that really serves as a starting point for this study. In this article, a normal form is presented and it is proved that it allows an efficient equivalence procedure. This normal form is based on the notion of earliest production, which is essentially a generalisation of the normal form for deterministic word transducers [Cho79].

Starting from there, we defined a notion of residual transformation that fits this normal form, and which is based on the notion of **canonical origin**, which is the topmost node that is responsible for the production of an output node. This notion allows us to define a Myhill-Nerode like theorem, with its associated construction. This construction allowed us in turn to define a learning algorithm.

Finally, in the case of top-down, the domain plays an important role as the transducer can ignore subtrees of its input. To verify that those subtrees fit a specific domain, the solution proposed in [EMS09] is to associate a tree automaton - called an inspection automaton - to the transducer to check the validity of input trees. This has serious consequences, first studied in [EMS09], but that we further explored in [BLN16].

Presentation of *DTOPs*

Tree transducers are essentially generalisations of tree automata that produce a tree at the same time as they run through it. As such, several good properties of word automata are carried over. Among all the formalisms on tree transducers, we focus on **deterministic top-down tree transducers**, or *DTOPs*.

Top-down tree transducers are essentially top-down tree automata with an output. They have as usualy a finite set of states, and their rules are of the following form:

$$q(f(x_1, x_2)) \to C$$

Here, q is a state, x_i are variables, f a binary symbol of the input tree, and C is an output tree that has some leaves labeled by some $q'(x_i)$ which indicate how to continue the transformation, and on which subtree. The transducer also has an axiom which is also such an output tree with some leaves labeled $q'(x_0)$, and which is output at the beginning of the transformation. Instead of giving a formal definition, we illustrate their behavior in the following example.

Example 7.

Consider the transducer defined the following way:

Axiom : $F(q(x_0), q(x_0))$
Rules :
$q(f(x_1, x_2)) \to F(q(x_1), q(x_1))$ $\qquad$ $q(a) \to A$

This transducer takes as an input trees built with only f and a symbols. It outputs a balanced tree built with F and A of height equal the leftmost branch of the input tree plus one.

Consider for instance the input tree $f(a, f(a, a))$. Using the axiom, the transducer first outputs an F that has as children two transformations of the tree with the state q. The state q then checks the first f, outputs a new symbol F with two transformations of a (the left child of the tree) by the state q. Finally, it transforms a into A, and outputs the tree $F(F(A, A), F(A, A))$.

Earliest Normal Form

Although not part of our works, we start with a quick presentation of results of [EMS09] that allow to define the normal form that we will need.

A *DTOP* is **earliest** if it produces as much as possible of its output *as soon as possible*. This corresponds to a syntactic property: for each state, the common context of each of its possible output is empty (otherwise, it could have been produced earlier).

Example 8.

Consider the following transducer :
Axiom : $q(x_0)$
Rules :
$q(f(x_1, x_2)) \to g(f(q(x_1), q(x_2)))$ $\quad$ $q(a) \to g(a)$

This transducer adds an additional g symbol upon every input symbol. This transducer is not earliest, as a g symbol is always output at the top of the output tree. Hence, it could already be produced in the axiom. This would also shift the production for the other g symbols. This results in the following earliest *DTOP*:

Axiom : $g(q(x_0))$
Rules :
$q(f(x_1, x_2)) \to f(g(q(x_1)), g(q(x_2)))$ $\quad$ $q(a) \to a$

Note that, in the first transducer, the output of the state q always starts with a g, while in the second transducer, there is no output common to every output of the state q. This is a property that all states of earliest transducers possess.

One good thing with the notion of earliest *DTOP* is that every *DTOP* can be transformed into an earliest one, hence we do not lose expressiveness. The obtained earliest transducer has the same number of states. Note however that the size of rules can be exponentially bigger.

Every earliest transducer can be made minimal with a polynomial time procedure by simply merging states that perform the same transformation, and there is a unique minimal earliest transducer for every transformation that can be described by a *DTOP*.

This has many consequences. First, this allows for an efficient equivalence procedure for earliest *DTOPs*. Second, that means that there is a canonical way to associate each output node to an input node: the one responsible for its production in any earliest *DTOP*. This last point will be of particular interest in our study, and will allow to define a Myhill-Nerode theorem.

A Myhill-Nerode Theorem

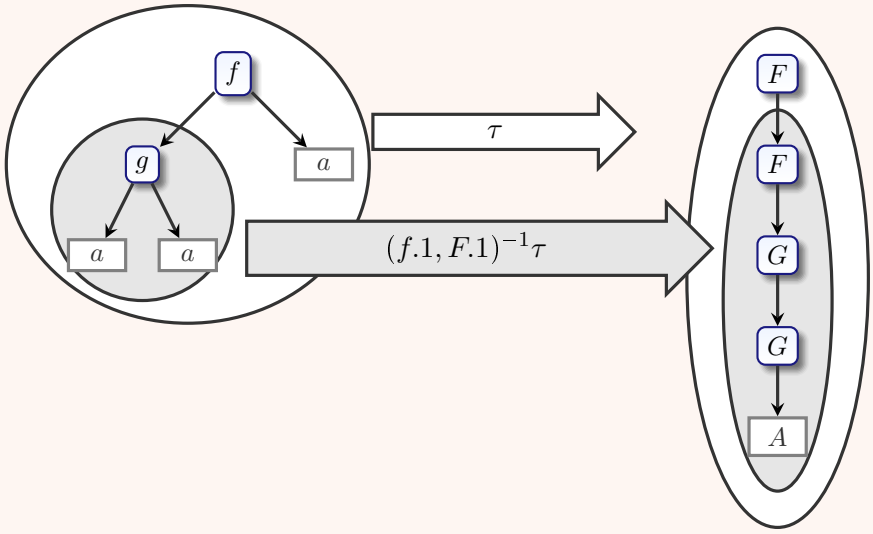
Above results have served as a basis for a Myhill-Nerode theorem we presented in [LMN10b]. To obtain this, we first need to define a proper notion of residual transformations. This will be defined on pair of paths.

First, for a path u and a tree t that contains u, we can define $u^{-1}t$ as the subtree of t that is under the path u. From this notion, a pair of input

/ output paths $(u; v)$ can define the residual of a tree transformation τ, denoted $(u; v)^{-1}\tau$. For a tree t, it transforms any subtree that appears in a tree t under the path u into the subtree that appears under the path v in $\tau(t)$. In other words, it transforms $u^{-1}t$ into $v^{-1}\tau(t)$. This is illustrated by the following example.

Example 9.

Consider the transformation τ that takes a tree composed of binary symbols f and g with a leaves, and transforms it into a tree composed of monadic symbols F and G with A leaves. It copies the left branch of the input tree, replaces symbols with their upper case corresponding and doubles every symbol. For instance, it transforms $f(g(a, a), a)$ into $F(F(G(G(A))))$.



The transformation $(f.1, F.1)^{-1}\tau$ is the function that transforms any subtree that appears under the path $(f.1)$ of a tree t (such as $g(a, a)$ that appears in $f(g(a, a), a)$) into the subtree that appears under $F.1$ in the image of t. For example, $g(a, a)$ is transformed into $F(G(G(A)))$ by $(f.1, F.1)^{-1}\tau$.

Note that not every pair of paths defines a residual transformation. For instance, the following residual transformations are not properly defined:

- $(f.1, G.1)^{-1}\tau$ is empty as G is never in the output of an input tree starting with an f,
- $(f.2, F.1)^{-1}\tau$ is not functional, as what appears in the output under $F.1$ does not depend on what is below $f.2$ in the input.

In fact, there is potentially an infinite number of distinct residual transformations for a transformation. However, only a finite subset of residual transformations is relevant for the Myhill-Nerode construction.

First, we naturally restrict ourselves to residual transformation $(u; v)^{-1}\tau$ that are **non-empty**. This means in fact that the path v should actually be part of the output of trees that contain u. In above example, the pair $((f.1); (G.1))$ is not valid as G is never produced by the transformation of a tree that has an f at its root.

Second, there are some badly matched pairs $(u; v)$ that can be recognised by the fact that $(u; v)^{-1}\tau$ is not **functional**. In our example, $((f.2); (F.1))^{-1}\tau$ is a non functional transformation, as what is under $(F.1)$ in the output does not actually depend of what is below $(f.2)$ in the input. This means that $(f.2; F.1)^{-1}\tau$ is not properly defined.

Third, those two restrictions still define a potentially infinite number of residual transformations. In fact, they correspond to every possible transformations defined by states of a transducer representing τ. We can restrict to only the transformation that correspond to states of an *earliest* transducer by considering only residuals $(u; v)^{-1}\tau$ for which v is **maximal**, i.e., such that there are no other residual transformations of the form $(u; v \cdot v')$ with a non-trivial v'.

In our example, the transformation $(f.1, F.1)^{-1}\tau$ is perfectly defined, but it is not maximal as it always output trees that starts with an F. The corresponding maximal residual transformation $(f.1, F.1.F.1)^{-1}\tau$

From this, a pair of paths $(u; v)$ defines what we call a proper residual of a transformation τ if it has those three properties, i.e. if it is **non-empty**, **functional** and **maximal**. We have two important properties: first, a transformation that can be represented by a $DTOP$ has a **finite** set of distinct proper residuals, and second, one can build a transducer we call $MIN(\tau)$ from this set. The construction of $MIN(\tau)$ is defined such that there is a one-to-one correspondence between states of $MIN(\tau)$ and the set of distinct proper residuals of τ, and the rules are defined using the earliest property. It turns out that this $DTOP$ is the minimal earliest $DTOP$ for τ. This allows us to obtain the following Myhill-Nerode-like theorem:

Theorem 4 ([LMN10b])**.**

For a tree transformation τ, the following are equivalent:

- τ is definable by a $DTOP$.
- τ has a final amount of distinct proper residual transformations.
- $MIN(\tau)$ is the minimal earliest transducer that describes τ.

Learning Algorithm for DTOPS

The Myhill-Nerode construction can serve as a basis for a learning algorithm, that we first defined in [LMN10b] and that takes aspects of both *RPNI* for trees [OG93] and *OSTIA* for word transducers [OGV93], along with its own specificities. The idea is essentially to build iteratively all states and all rules of the minimal earliest *DTOP*, using a finite sample of pairs of input / output trees to estimate the operations needed in the construction. Like our other learnability results, this guarantees to converge toward a minimal earliest transducer if the sample has good properties, and we proved that such samples of reasonable size exist.

We illustrate this by an example. Consider the transformation τ that takes some binary trees and outputs their mirror, replacing each symbol by its upper case. We can infer it from the sample containing the following pairs of trees $f(a,a)/F(A,A)$, $f(a,b)/F(B,A)$, $f(b,a)/F(A,B)$, and $f(f(a,b),a)/F(A,F(B,A))$, as well as examples of trees with just one leaf, i.e., a/A and b/B.

Axiom Building First, as there is no common output for output trees, we start with an axiom containing only the state associated to $(\varepsilon;\varepsilon)$ that we denote $[\varepsilon,\varepsilon]$. So the axiom is $[\varepsilon,\varepsilon](x_0)$.

Rule Building We then build rules for the state $[\varepsilon,\varepsilon]$. For constant symbols a and b, we simply encode the identity function, for instance $[\varepsilon,\varepsilon](a) \to A$, as we observed the pair a/A in the sample. For the binary symbol f, we observe that the output of every tree of the form $f(t_1,t_2)$ always starts with an F. This allows to build a rule of the form $[\varepsilon,\varepsilon](f(x_1,x_2)) \to F(q(x_i),q'(x_i'))$. The remaining question is how to define states q and q', and what are the values of i and i'.

To mimic the Myhill-Nerode construction, states q and q' should both correspond to a pair of output / input paths. The output paths are respectively $(F,1)$ and $(F,2)$, which are the paths that reach the position of the corresponding states in the right hand side. The input path should be of the form $(f.j)$ and $(f.j')$. And there is only one possible value for j and for j' which is given to us by the **functionality**. For instance, $(f.1;F.1)^{-1}\tau$ is not functional, and this is observed by the output of $f(a,a)$ and $f(a,b)$. So we have to consider only the possibility of $j=2$ and $j'=1$. In the end, we build the rule

$$[\varepsilon,\varepsilon](f(x_1,x_2)) \to F([f.2;F.1](x_2),[f.1;F.2](x_1))$$

New States We considered two new pairs of paths $(f.1;F.2)$ and $(f.2;F.1)$, which could potentially correspond to new states. There is however the possibility that those two new pairs correspond to already existing states. This

is obviously what happens here, and that is confirmed by the fact that, if we conjecture $((f.1);(F.2))^{-1}\tau$ based on the sample, we obtain a/A, b/B and $f(a,b)/F(B,A)$, which does not contradict our estimation of $(\varepsilon;\varepsilon)^{-1}\tau$ (which is simply the whole sample). If we had only one pair that would contradict, we would have known that this pair corresponds to a new residual transformation (meaning we need to build a new state). Here, we simply replace the state $[(f.1);(F.2)]$ by $[\varepsilon,\varepsilon]$. In the end, we obtain the rule $[\varepsilon,\varepsilon](f(x_1,x_2)) \to f([\varepsilon,\varepsilon](x_2),[\varepsilon,\varepsilon](x_1))$

In order to make this algorithm infer the correct transducer, the sample needs to contain pairs of trees that allow all the hypothesis that we made to be correct. For this, we need an amount of pairs of trees which is quadratic in the number of states of the minimal earliest transducer. This gives us the following theorem:

Theorem 5 (Learnability of *DTOP* [LMN10b])**.**

The class of transformation represented by *DTOP* are learnable from polynomial time and data in the following sense:

- there exists a learning algorithm $learner_{DTOP}$ such that from any sample S, $learner_{DTOP}(S)$ provides in polynomial time a *DTOP* A consistent with S
- for any *DTOP* A describing a transformation τ, there exists samples S_A of cardinality polynomial in the size of A such that, for any sample S consistent with τ that contains S_A, $learner_{DTOP}(S)$ is a representation of τ

Domain Issues

In the case of *DTOPs*, taking into account the domain causes several problems. The first problem is that, unlike other formalisms of transducers, such as word transducers, *DTOPs* present the particularity that they ignore parts of the input tree which are not transformed. As a consequence, they may not always check their own domain by themselves.

Imagine for example a transformation that would keep only the left branch of any tree, but operates only on trees whose right branch is of even height. A *DTOP* would simply erase all branches except the first one, and has no therefore no way to verify the height of the right branch.

A first solution has been proposed in [EMS09]. It consists in coupling the transducer with a tree automaton called **an inspection** that simply checks whether the input tree is in the domain or not.

The second problem is that, as *DTOP* can perform copies, there may be several states that transform the same subtree, potentially with different domains. Consider for instance the following example:

Example 10.

The following transducer uses two states q_1 and q_2 that both perform identity on string-like trees with monadic symbols f, g and h, with a leaves. The state q_1 accepts f and g symbols while the state q_2 accepts f and h symbols. The axiom actually uses both states on the same input tree, which make the whole transduction defined only for monadic trees built with the f symbol.

Axiom: $f(q_1(x_0), q_2(x_0))$

Rules:

$q_1(f(x_1)) \to f(q_1(x_1))$ $\quad$ $q_1(g(x_1)) \to g(q_1(x_1))$ $\quad$ $q_1(a) \to a$

$q_2(f(x_1)) \to f(q_2(x_1))$ $\quad$ $q_2(h(x_1)) \to h(q_2(x_1))$ $\quad$ $q_2(a) \to a$

Note that allowing this kind of behavior can sometimes reduce the size of the transducer. Imagine for instance a transducer that would perform two copies of its input tree (under a dummy root), the first copy could recognise only trees of even height while the second copy recognise trees of height which is a multiple of 3. Another equivalent transducer that would recognise trees of height which is a multiple of 6 on both its branches would be bigger, as it would require 6 states instead of the $(2 + 3 =)5$ states of the first transducer.

However, allowing this kind of trick to reduce the size of the transducer also prevents to define a minimal earliest normal form as before. The solution is to forbid this kind of behavior. For *DTOP* with inspection, that corresponds to forbid to have states that can provide an output which is not defined in their domain (as obtained from the inspection).

[EMS09] presented this solution for domains defined by top-down automata. This works quite naturally as the inspection is itself top-down, and hence, the domain of states can be easily defined that way by associating each state of the transducer to a unique state of the inspection.

In [BLN16], we extend this solution for the case of regular domains, i.e. domain defined by tree automata which are not necessarily top-down. The main problem that is caused is that there are now several input nodes that could be the origin of an output node. Consider the following example.

Example 11.

Consider the following simple transformation, consisting only of those two pairs of trees built with the binary symbol = and two leaf symbols 0 and 1. Note that the domain of this transformation is not regular.

$= (0, 0) \to 0 \qquad = (1, 1) \to 1$

Both following transducers are valid for this transduction:

Axiom: $q(x_0)$
Rules:
$q(= (x_1, x_2)) \to q'(x_1) \qquad q'(0) \to 0 \qquad q'(1) \to 1$

And:

Axiom: $q(x_0)$
Rules:
$q(= (x_1, x_2)) \to q'(x_2) \qquad q'(0) \to 0 \qquad q'(1) \to 1$

The first transducer simply copies the first symbol, while the second transducer copies the second symbol. In both cases, those input symbols can be considered as a valid origin for the output symbol, as both symbols are equal as specified by the domain.

Essentially, the solution proposed in [BLN16] consists in picking the leftmost choice (for instance). In this case, that would mean considering the first transducer as the canonical one. In the above example, this is trivial, but note that this is not always the case, as illustrated by the following example.

Example 12.

Consider the same transformation as in the previous example, except that the left branch of the = is the tree representation of a simple logical Boolean formula whose evaluation is in the right branch. This transformation contains for instance:

$= (\wedge(0, 0), 0) \to 0 \qquad = (\vee(0, 1), 0) \to 0$
$= (\vee(\wedge(0, 1), 1), 1) \to 1$

As the transformation that evaluates such a logical formula can not be performed by a $DTOP$, the only valid $DTOP$ that performs this transformation is the one that considers only the right branch:

Axiom: $q(x_0)$
Rules:
$q(= (x_1, x_2)) \to q'(x_2) \qquad q'(0) \to 0 \qquad q'(1) \to 1$

Intuitively, to obtain a normal form, we chose to consider, at each level, the left-most possibility that can be performed by a *DTOP*. Exact details can be found in [BLN16].

4. Contribution: Tree to String Transducers

Contribution

Results presented here are part of the PhD thesis of Gregoire Laurence [Lau14] and have been published in [SLLN09, LLN+11, LLN+14].

Our second contribution in the domain of transducer learning concerns the class of tree to string transducers. This class naturally corresponds to some applications, such as XML to LaTeX conversion for instance, but note it is also in general more expressive than tree to tree transductions as the output can be the serialisation of a tree. They also serve as an interesting baseline for the study of more complex class of transducers such as Macro Tree Transducers, as they share some common mechanisms. In particular Macro Tree Transducers can encode tree to string transducers.

Presentation

Our first contribution is the definition of this class. Even though it is a rather natural class, we are not aware of other similar works. The class that can be considered as the closest is maybe the class of **visibly pushdown transducers** - or VPT [FRR+10] - which read serialisation of trees (as a visibly push-down automata [AM09]) to produce a string. However, the purpose is different - VPT focus more on a *streaming* reading of the tree - and the reading order of the input is different, as VPT operates in a depth-first search.

We introduced the model of **Tree-to-string transducers** in [SLLN09]. It mixes a top-down tree automaton for the input with a context-free grammar for the output. Rules are of the following form:

$$q(f(x_1, \ldots, x_n)) \to w_1 \cdot q_1(x_1) \cdot w_2 \cdot q_2(x_2) \cdot w_3$$

where q are states, x_i are variables, and w_i are substring of the output alphabet. The global output is composed of strings u_i concatenated together. In the above example rule, the input is checked in the natural order, and every input subtree is checked once, but we can also imagine tree-to-string transducers which can perform copies, or which change the order of the output. We call a tree-to-string transducer **linear** if it has no copy (every x_i

appears at most once in the right hand side of a rule), and **sequential** if it is linear and if the order is preserved (the variables x_i on the right hand side are always ordered from x_1 to x_n).

We illustrate this on the following example.

Example 13.

Axiom: `START` $\cdot\, q(x_0) \cdot$ `END`
Rules:
$q(f(x_1, x_2)) \to$ `<f>` $\cdot\, q(x_1) \cdot q(x_2) \cdot$ `</f>` $\qquad q(a) \to$ `<a/>`

This tree-to-string transducer performs an XML-like serialisation of a tree, and encompasses it between a `START` symbol and an `END` symbol. For example, the tree $f(f(a, a), a)$ is transformed into `START <f> <f> <a/> <a/> </f> <a/> </f> END`
This transducer is sequential, as every input symbol is checked only once, and the order is preserved.

Tree-to-string transducers are suited to deal with transformations such as XML to LaTeX or other textual formats, but they can actually capture a subclass of tree transformations, as the output string can be the serialisation of a tree.

They can also be seen as a specific case of Macro Tree Transducers as a tree-to-string transformation can be encoded into a tree to tree transformation where the output tree is monadic (i.e. a tree composed only of monadic or constant symbols). This means that they are a good study case before dealing with macro tree transducers.

Earliest Normal Form

After defining the class, we applied our methodology on this class in order to find a **normal form**, a **Myhill-Nerode-like theorem**, and a **learning algorithm**.

The normal form is based - again - on the idea of producing *as soon as possible*. This takes however an unusual aspect on this class. The earliest normal form is indeed based on two notions. First, the output string needs to be output as *up* as possible in the transducer. This notion of *earliest* is very similar to the one we described for *DTOPs*, however, this is not enough to define a normal form. Observe this on the following example:

Example 14.

Axiom: $q(x0)$
Rules:
$q(f(x_1, x_2)) \to q(x_1) \cdot q(x_2) \cdot \#$ $\qquad$ $q(a) \to \#$

This transducer takes as an input a tree build with f and a symbols, and outputs a string of # symbols whose length is equal to the number of symbols in the input tree.

This transducer is not earliest in any sense. For instance, whenever a binary symbol is read, at least two # are always produced by the transformation. However, the transducer only produce one of the symbol (the second is producer later). An *earlier* transducer would be the following one:

Example 15.

Axiom: $q(x0) \cdot \#$
Rules:
$q(f(x_1, x_2)) \to q(x_1) \cdot q(x_2) \cdot \#\#$ $\qquad$ $q(a) \to \varepsilon$

An earlier transducer that performs the same transformation. The production of # has been shifted up. For instance, the axiom already produces one # symbol now.

This transducer is **upmost** earliest in the sense that the production of # symbols is done by a rule as soon as possible. However, the rule itself is not earliest as the # can be produced sooner (or more on the left) in the rule, as can be seen by this last transducer :

Example 16.

Axiom: $\# \cdot q(x0)$
Rules:
$q(f(x_1, x_2)) \to \#\# \cdot q(x_1) \cdot q(x_2)$ $\qquad$ $q(a) \to \varepsilon$

The production of this transducer has been shifted as left as possible in every rules. The # symbols are now produced at the beginning.

A transducer like this - that produces its output as left as possible in the rule - is called **leftmost** earliest. And we define a tree-to-string transducer as **earliest** if it is both **upmost** and **leftmost** earliest. The notion of earliest leads to a normal form: we state in [LLN+11] that every sequential tree-to-word transducer has a unique minimal earliest equivalent tree-to-string transducer. We call this transducer the canonical tree-to-string transducer.

A Myhill-Nerode Theorem

From this normal form, we aim to define a Myhill-Nerode theorem, and in particular a direct construction of this normal form. As for our other formalisms, this starts with a definition of **residual transformations**. The definition we use is a residual transformation relative to a path p for a transformation τ that we denote $p^{-1}\tau$. We use a recursive definition for the **canonical transducer**, that we define conjointly toa **Myhill-Nerode construction**.

The first step consists in building the **axiom** and the residual $\varepsilon^{-1}\tau$. This is done by first computing the largest common prefix u and the largest common suffix v of all the output of τ. The axiom is then $u \cdot q_\varepsilon(x_0) \cdot v$, where q_ε is a state that will define the residual $\varepsilon^{-1}\tau$. This is defined by striping u and v out of the output of τ. Note that this mean that in general $\varepsilon^{-1}\tau$ is different from τ.

Example 17.

For the transformation defined in previous examples, every output string starts with an `#` and ends with an `#`. As it can actually be the same `#`, it is not counted twice and we use the axiom $\texttt{\#} \cdot q_\varepsilon$. For the same reason, $\varepsilon^{-1}\tau$ is defined by taking τ and removing one `#` symbol from the output.

Rules and the other residual transformations are then defined recursively alongside with rules of the transducer. So, if a path p is associated to a state q_p and a residual $p^{-1}\tau$, we create rules of the form:

$$q_p(f(x_1, x_2)) \to u_0 \cdot q_{p\cdot(f.1)}(x_1) \cdot u_1 \cdot q_{p\cdot(f.2)}(x_2) \cdot u_2$$

where for instance the state $q_{p\cdot(f.1)}$ is associated to the newly defined residual $(p\cdot(f,1))^{-1}\tau$. All elements of rules are created by another recursion. First, u_0 is the largest common prefix of every output string of $p^{-1}\tau(t)$ for trees t starting with an f. Next, $((p\cdot(f.1))^{-1}\tau)(t)$ is obtained by considering all trees of the form $t' = f(t, t_2)$ and their corresponding output string $p^{-1}\tau(t')$, by removing u_0 on the left, and removing on the right all the

strings that do not change when t is replaced by another subtree. The exact procedure is detailed in [LLN+11], and leads to the recursive definition of all $p^{-1}\tau$, and of the canonical transducer of τ, if one also merges equivalent states.

We illustrate this on the following example.

Example 18.

Following the previous example, from q_ε, we can build the following rules. First the rule $q_\varepsilon(a) \to \varepsilon$ comes from the fact that $\varepsilon^{-1}\tau(a) = \varepsilon$.

The rule $q_\varepsilon(f(x_1, x_2)) \to u_0 \cdot q_{(f,1)}(x_1) \cdot u_1 \cdot q_{(f,2)}(x_2) \cdot u_2$ is then created. The string u_0 is instantiated by `##` as the output by $\varepsilon^{-1}\tau$ of every tree whose root is an f contains at least two `#` symbols.

We then define $(f,1)^{-1}\tau(t)$ for each possible t. For instance, $(f,1)^{-1}\tau(a)$ is obtained by considering the largest common prefix of $\varepsilon^{-1}\tau(f(a,t'))$ for every possible tree t'. This is `##`, from which we remove u_0, to get ε. Actually, we can realise that $(f,1)^{-1}\tau = \varepsilon^{-1}\tau$ and we can use q_ε instead of $q_{(f,1)}$. Similarly, we can compute $u_1 = u_2 = \varepsilon$, and $(f,2)^{-1}\tau = \varepsilon^{-1}\tau$.

Ultimately, this results in the creation of the rule $q_\varepsilon(f(x_1, x_2)) \to \texttt{##} \cdot q_\varepsilon(x_1) \cdot q_\varepsilon(x_2)$.

Learning Algorithm

As our methodology suggests, the **learning algorithm**, described in [LLN+14], is obtained by mimicking the above construction on a finite sample. In order to make correct assumptions on residual transformations, it needs a sample that contains every example tree that allows the perfect computation of $p^{-1}\tau$, for all p that are the smallest representative of their equivalence class (i.e. for which there is no smaller p' with an equivalent residual transformation).

As a difference with the other learnable classes of transductions presented here, a difficulty arises with tree-to-string transducers. Typically, we require a certain robustness of the learning algorithm in the sense that, even with an imperfect sample, we are still capable of providing a transducer that is at least consistent with the sample. This is not possible in general here as there is no consistency algorithm for the class of tree-to-string transducers: the simple problem to provide a tree-to-string transducer compatible with a set of pairs of input trees / output trees is NP-complete [LLN+14].

As a consequence, we can not guarantee that our polynomial time learning algorithm always provides a tree-to-string transducer which is consistent with the set of pairs of input tree / output string it receives as an input.

This situation can however be detected easily, and interpreted as a failure case of the algorithm, which simply requires more example.

Note that all above results has been done on sequential tree-to-string transducers. It has been extended since then by Boiret and Palenta in [BP16] to the linear case.

5. Contribution: Transducers with Look-Ahead

Contribution

Results presented here have been published in [BLN12] (word case only)

For a transducer, a **look-ahead** is a companion deterministic automaton that performs a first run on the input, reading the input in a direction different from the one used by the transducer, and annotates positions of the input with its states. In the tree case, it can be for instance a deterministic bottom-up tree automaton that helps a top-down deterministic tree transducer [Eng76]. We call $DTOP^R$ the class of $DTOP$ with regular look-ahead. This allows to increase the expressiveness of the class while not changing much of the computational cost of performing the transformation. In fact, it allows to obtain the same expressiveness as non-deterministic functional top-down tree automata, while keeping a polynomial time algorithm for computing the output of an input tree.

In the string case, a deterministic word transducer with regular look-ahead is a deterministic transducer (also called a sequential transducer) coupled with a deterministic automaton that performs a first pass on the input string right-to-left. This formalism allows to capture the whole class of word rational functions, i.e. the class of functions that can be expressed by non-deterministic word transducers[EM65] or by monadic second order logic on words [Cou94].

However, the look-ahead adds some difficulties. The main one is that it becomes harder to define a proper normal form, and therefore, to establish learning results. We attempt to lift those difficulties, first on the word case. Note that the new difficulty comes only on how to define a normal form for the look-ahead itself. This is because, once this is obtained, it becomes simple to use results on transducers without look-ahead by just decorating input trees with states of the look-ahead, and consider them as the input of the transducer.

Normal Form

The seminal work of [RS91] provides a construction that gives a normal form for the class of rational functions, which corresponds to the class of transformation performed by functional words transducers (not necessarily deterministic). This normal form however is defined for the class of bimachines, an alternative representation for word rational functions, but bimachines can be transformed easily into word transducers with look-ahead.

A **bimachine** [Eil74] is a combination of two deterministic word automata : one that reads an input word left-to-right and another that operates right-to-left, and a function that, given a pair of states of each automaton and an input symbol, provides an output subword. Each position of the input word is associated to two states, one that corresponds to the reading of its prefix by the first automaton, and one for its suffix, and the function gives the corresponding output of the letter found at the current position. The transformation operated by a bimachine on the whole word is obtained by concatenating the sequence of each of those partial transformations operated at each position of the input word.

The result of [RS91] consists in first establishing a normal form for the right-to-left automaton, and then, use this automaton to obtain the left-to-right automaton and the rest of the bimachine.

We studied this model in [BLN12] and adapted those results to formalisms relevant for our approach. Our first result consisted in pointing out that there is a strong link between **bimachines** and **deterministic word transducers with look-ahead**. In fact, the right-to-left automaton can be easily converted into a look-ahead, while the rest of the machine allows to obtain a deterministic transducer. The main consequence for us is that the normal form for bimachines allows us to define a normal form for deterministic transducers with look-ahead.

Myhill-Nerode theorem

The result of [RS91] allowed us to adapt the normalisation result for bimachine in order to obtain a **normal form for the look-ahead** of a transducer with look-ahead. This is based on the idea that two possible suffixes of a word are equivalent (and hence correspond to the same state of the look-ahead) if adding them to the end of a word does not alter the output much. *Alter much* essentially means here that the possible set of alterations is finite, but this is best understood on an example.

Consider for instance a transformation τ that copies the last letter of a word over the alphabet $\Sigma = \{a, b\}$ to the first position, transforming for instance *abbab* into *babbab*. This can not be performed by a deterministic word transducer, but can be done easily by a deterministic word transducer with a look-ahead. Consider the suffix '*ab*' and the suffix '*bb*', we define

τ_{ab} as the operation that transforms a word u into $\tau(u.ab)$ and τ_{bb} defined similarly. Both essentially add a b symbol at the beginning of the word and ab or bb at the end. The difference in term of suffix between $\tau_{ab}(u)$ and $\tau_{bb}(u)$ is always ab and bb, i.e. a finite set.

Consider now $\tau_{ab}(u)$ and $\tau_{ba}(u)$, the difference between both words is composed of all possible pairs of $b.u.ab$ and $a.u.ba$, i.e. an infinite set. This allows us to say that ab is equivalent to bb (and should correspond to the same look-ahead state), while ab and ba are not.

From the equivalence relation based on this idea, one could define properly a normal form for the look-ahead, much as one would define the minimal normal deterministic automaton: two equivalent suffixes go in the same state of the look-ahead. In other words, the states of the look-ahead each correspond to an equivalence class of the transduction. From this, one could define the corresponding (classical) deterministic word transducer as in [Cho79].

Two important points are to be noted here. First, **the transducer is dependant on the choice of the look-ahead**. In particular, our definition allows us to define a minimal look-ahead, but that does not mean that the transducer is minimal: it is minimal with respect to this particular look-ahead, but it is possible that a bigger - and more informative - look-ahead allows to define a smaller transducer. This also means that the overall size of the transducer with look-ahead (adding size of both elements) has no reason to be minimal in this construction, and can even be sometimes exponentially bigger than some other equivalent transducer.

Second, the construction that allows to define the minimal look-ahead is based on **checking whether a particular set is finite or not**. This is a problem for the machine learning point of view, as when we have as an input only a finite sample of the whole transformation, this can obviously not be observed directly.

Learning Algorithm

The final result we obtained in [BLN12] was to establish a polynomial time learning algorithm for the class of deterministic transducers with regular look-ahead that is based on the above two points. The algorithms has three main components.

The **first component** infers a look-ahead. It has two integer parameters m and l. Remember that building the look-ahead corresponds to building an equivalence relation between suffixes, and two suffixes are equivalent if a particular set built on them is finite. The integer m sets the limit size from which the arity of this set is considered infinite, and thus establishes that that two suffixes are not equivalent. If m is too small, the equivalence relations get a too big index, which results in a look-ahead which is bigger

than the minimal one. This is not a problem in principle, but if we can expect the input sample to be characteristic for the minimal look-ahead (in the sense that it contains all examples that allows a proper estimation of its equivalence relation), we can not expect it to be characteristic for arbitrarily big look-ahead. The integer l sets the maximum size of the look-ahead. If the built look-ahead has more than l states, the look-ahead is simply rejected, and we have a case of failure.

The **second component** iterates on all the pairs (m, l) in a somehow diagonal way : $(1, 1)$, $(2, 1)$, $(2, 2)$, $(3, 1)$, ... For each pair (m, l), it calls the first component with these parameters. The procedure stops for the first value (m, l) which does not fail, i.e., for which we obtain a look-ahead of l states. Our result is that, for a sample characteristic for the minimal look-ahead, the algorithms indeed stop for a certain pair (m, l) and provides a look-ahead for the target transformation. This look-ahead is not necessarily the minimum one, but the diagonal traversal ensures that its size is not more than twice the size of the minimal one.

The **third step** consists in using the classical learning automaton for deterministic word transducer **OSTIA** [OGV93] on words decorated with states of the inferred look-ahead. In the end, the learning algorithm ensures a polynomial learning of rational functions, with the particularity that it does not necessarily provide the canonical transducer.

Extension to the tree case

The extension of above results to the **tree case** of this approach is still an open problem. The relation of equivalence used to define the look-ahead can be translated to the tree case, but it is not enough to define a *tree* look-ahead. In fact, one can see that for any $DTOP^R$, the equivalence relation induced by the look-ahead (two trees are equivalent if they correspond to the same look-ahead state) is necessarily a refinement of the tree extension of the Reutenauer-Schützenberger relation. However, the reverse is not true.

The natural extension of the results above to the tree case leads us to consider deterministic top-down tree transducers that uses a deterministic bottom-up automaton as look-ahead [Eng76]. This class, called top-down tree transducers with regular look-ahead, or TDLA, has the same expressiveness as non-deterministic top-down tree transducers. The extension of results for the word case is however still a work in progress and presents some unexpected difficulties that we quickly highlight here.

First, the equivalence relation of Reutenauer-Schützenberger can be extended without much difficulties to the tree case. However, this new equivalence relation does not directly gives us a look-ahead. It remains that this equivalence relation is important. In fact, if we consider any TDLA that represents a transformation τ, two subtrees that evaluate to the same state

of the look-ahead are necessarily equivalent with respect to this equivalence relation.

However, this is not sufficient to define a proper look-ahead. One reason for this is that, due to the nature of trees, there are subtrees which generate only a finite difference in the production, but can not be considered equivalent for the look-ahead.

Consider for instance the simple transduction that evaluates Boolean expressions (for instance $\wedge(\vee(1,0),1)$ is transformed into 1). As the output space of this function is finite, any subtree would be considered equivalent. If we follow strictly the same approach as for the word case, this would result to define a trivial 'one-state' look-ahead. However, this transformation can not be done by a *DTOP* without look-ahead.

Another difficulty comes from the fact that, unlike the word case, there is no unique minimal look-ahead for a transformation. Consider now the transformation that transforms $f(a, A)$ and $f(b, B)$ into 0 and $f(a, B)$ and $f(b, A)$ into 1. There are two minimal look-aheads for this transformation. The first one verifies the label of the left leaf, and does not care about the right leaf - A and B are sent to the same state. The top-down transducer then explores only the right leaf and can decide the output thanks to the information from the look-ahead. The other look-ahead does the same, but with the right leaf.

This means we will need to refine our definition of equivalence relation that would allows to define a look-ahead, with a need to take into account both those problems. This is still a work in progress.

6. Toward Learning MSO Transformations

We presented here three examples of classes of transformations for which we defined a learning algorithm, following our methodology. As we mentioned earlier, they all converge toward the inference of macro tree transducers with regular look-ahead, a class stronger than MSO definable transformation, but equivalent when one adds the restriction of being essentially copy bounded (technically, this corresponds to the *single-used restriction* defined in [EMS09]).

The inference of *DTOPs* offers a base case, as they are strictly more restrictive than MTT. Note however that we do not need to add the restriction of bounded copy to achieve learnability results for *DTOPs*.

Macro tree transducers are a complex model capable of many subtle operations. However, it is essentially a generalisation of tree to string transducers, as one can see the produced string as a simple tree, replacing string concatenation by vertical addition of subtrees. It seems therefore plausible to extend those results to MTT, although it is still an open question.

Probably, some restrictions will apply however. For instance, limitations that we had on tree-to-string transducers apply to MTT. This means that it will probably be hard to obtain learnability results with unbounded copy, at least on the vertical direction, as lifting this restriction would mean that it would be possible to have an inference algorithm for unrestricted tree-to-string transducers, which we think is unlikely.

Finally, we proposed learnability results for word transducers with look-ahead. It is still an open question whether those results can be ported in the tree case. Moreover, it is not clear how to combine all the mentioned above results to define learning algorithm for classes of transducers that have aspects of macro tree transducers and regular look-ahead.

To sum it up, results we presented each give one part of the solution. It is however doubtful that a normal form for the whole class of MSO definable transformations can be found. On the other hand, even with *DTOPs*, we capture transformations with unrestricted copies which are not MSO definable. We believe MTT with regular look-ahead is still a target of interest, giving a clear direction to our research. Most probably however, the best we can achieve is to obtain learnability for some subclass yet to be defined.

7. Conclusion

Works presented in this chapter have been the object of several publications. They all have already mentionned, but we sum it up here.

In [LMN10b], we introduced a Myhill-Nerode theorem for the class of top-down tree transducers, as well as an inference algorithm. We also proposed suggestions to use this approach in XML applications, with for instance, an ad-hoc encoding of XML trees.

Those results has been extended in [BLN16] to deal with domain issues. In particular, solutions to deal with regular domains, and not only top-down domains, have been studied.

We started the study of tree-to-string transducers with [SLLN09], where we studied the equivalence problem. In particular, we presented a polynomial reduction between the equivalence problem of this class and the equivalence problem of other classes, such as nested-word to word transducers.

The study of tree-to-string transducers has been pursued in [LLN+11], where we established a normal form and an inference algorithm for the class of sequential tree-to-string transducers.

Finally, inference of rational functions, i.e., functional transformations performed by word tranducers, has been presented in [BLN12].

4 Graph Queries

While many semi-structured data formats can be formalised by trees, in many cases, the structure can be more complex, and graphs becomes the favoured formalisation. This is of course the case for data which are stored explicitly as graphs, such as RDF in semantic web. But this is also the case for data stored as trees, such as XML or JSON, but which contains references between elements, using a foreign key mechanism. For instance, in an XML document, the property 'author' of a book may contain only the identifier of an author, leaving all other data about him in another part of the document. This corresponds to an implicit link between different elements of the document.

Queries over tree data structures also commonly use this feature, either to link together several elements of the structure (*'give me sets of books with the same author'*), or to query the structure for a specific data value (*'give me all books of author x'*). The equality relation between the name of author in the query and in the data structure can also be seen as an implicit link. This means that the query actually uses graph features on the input tree.

The predominant query language for RDF graphs is SPARQL, a language inspired by SQL. However, since its version 1.1, SPARQL allows to use conjunction of regular path queries (CRPQs). A regular path query (RPQ) [CMW87a, CGLV02] is defined by a regular expression built on predicate labels. This includes in particular recursions, and one can ask for `(x isFriendOf+ John)` to obtain everybody linked by a friendship chain with John. An RPQ links therefore two elements of the graph. CRPQ are naturally conjunctions of such links, and allows to obtain more complex patterns.

The problematics we described in previous chapters for tree queries also apply to graph queries. In particular, they can be hard to write manually. This is particularly true as it is hard for a human to have a global comprehension of the structure of RDF graphs, as they tend to be quite big and complex in practice. Machine learning could also help to address this problem. To achieve this result, I propose to extend the inference methods developed for tree queries to the graph case.

In order to do this, we first observe that graph queries can be seen as a direct generalisation of tree queries. Consider for instance monadic graph queries, i.e. queries which select nodes in a graph. From a specific node, one can observe the tree unfolding of a graph. In the same way as a tree query can be defined by a tree language, a monadic graph query can be defined by a language of rooted graph, or by the language of its tree unfolding.

Some difficulties appear immediately with this approach. First, although they have in this case a finite representation, unfolding trees are **infinite** in general. This is not necessarily a major difficulty, as there exists finite state machines that deal with infinite objects. Büchi automata operate on infinite words (see for instance [Tho90]) and Rabin automata on infinite trees [Rab69]. In the word case, it is interesting to note that there exists learnability results for Büchi automata [dlHJ04].

However, for the same reason we needed a **pruning operation** for the inference of tree queries, a similar technique has to be developed for the graph case. This would solve several problems. Indeed, the main difficulty is probably that graph databases tend to have an important size, while the part relevant for the selection of a particular node is much smaller. This was already one major reason why we needed pruning in tree documents, but it is even more true in this case. This would allows us to come back to finite structures.

The second difficulty, probably harder to lift, is that unfolding a graph removes the information on its **cycles**. This information could however be preserved by adding an identifier to each node of the graph - as a **data value** - and using the same identifier for copies in the unfolding of the same node. For this specific problem, we propose to adapt techniques used to deal with data values for word automata, and that we detail a bit later.

Even using these different techniques, there are several important questions that remain. The most important one is certainly to establish what precise class of queries we can express with finite state machines such as the ones we describe (i.e. tree automata somehow enhanced with mechanisms we suggest) and which would be learnable. A nice goal is certainly to reach for MSO queries on graphs.

Note that all of this is essentially work in progress. In the following, we discuss a bit more precisely two aspects of this. First, we discuss techniques to find parts of the graph relevant to the query (the equivalent of pruning for trees). We start with a first restriction that deals with **path queries**, an already interesting subclass of queries which is used in practice. This has already led to some published content, that we quickly present.

We then present ideas on how to extend these works to the more general case of graph queries defined by **tree languages**.

After that, we discuss aboutf the problem of **data values**. Following works of Puppis [BLP10] on word automata, we present how we aim at extend this to the case of tree queries, a first step before reaching for the more general case of graph queries.

Finally, we conclude with some thoughts on how to extend this work on graph queries to the even more general case of graph transformations, as well as indicating how the elaboration of inference techniques for graph transformations can have an impact on the usage of databases.

1. Regular Path Queries

We start to study the problem of inference of graph queries by studying the inference of Regular Path Queries, or RPQs. In [BCL15b], we proposed preliminary works for this that we present here.

First, note that *classical* RPQs select pairs of nodes. An RPQ is defined by a regular expression, and a pair of nodes is selected by the RPQ if there exists a path between those nodes labelled by a word that fits the regular expression. We focused however on monadic queries, as this case is in fact harder to deal in machine learning. Here, a node is selected by a query if there exists a second node and a path between them labelled by a word that fits the regular expression.

For instance, consider the simple expression `isFriendOf+`. The binary query it defines selects pairs of nodes linked by a sequence of `isFriendOf` labels. The monadic query selects nodes which are first components of such pairs.

We studied two frameworks for the inference of monadic RPQs: a static one and an interactive one. First, the **static framework** consist in learning the RPQ from sets of nodes which correspond to the output of the target query, and similarly a set of counter-examples.

The inference is two-fold. First, for each example, we need to **identify the path** responsible of its selection. This is done using counter-examples. Indeed, each counter-example node describes a language of counter-example paths, in the sense that *none* of the paths that can be derived from a counter-example node can be in the target language. By contrast, each example node also defines a language of possible paths, for which at least one intersects with the target query. In fact, for each (positive) example, our algorithm picks the *smallest* path that it defines which is *not* part of a language defined by a counter-example.

With this approach, we obtain for each example node one path which is supposed to be responsible for its selection. Those paths, and the languages defined by counter-example, are used by a regular word language algorithm such as *RPNI*, only slightly altered. This allows us to have a learning algorithm able to infer RPQs.

Another important problem of this approach is that, to find paths for the examples, we basically need to compute an intersection of regular language represented by non-deterministic automata, a PSPACE-problem in general. To avoid this problem, we proposed several ideas. First, we can limit the length of paths. This of course cuts the complexity, and fits our preliminary experiments where typical paths are small, but is somehow unsatisfying as it also reduces the expressiveness of our algorithm.

The second solution comes with the **interactive framework** we want to use. In this setting, inspired by our interactive learning framework for tree queries [CGLN07] and by the Angluin learning model [Ang87b], an end-user may provide example nodes, or correct example nodes proposed by the learning algorithm. This can be achieved by plugging our static algorithm into this framework : the *static* learner infers hypothesis queries from examples or counter-examples obtained through interactions with the end-user.

This setting is certainly more realistic than the static setting, and seems to correspond more to what a *real-life* application would be. However, this causes new problems. In particular, the visualisations problems are less trivial than in the tree case: how can the user observe the graph, how to present examples to him... On the other hand, it also allows to imagine new ways to interact with the user. The objective being to find interaction which are still simple, but which are more informative than just corrections. In [BCL15a], we investigate for instance the use of interactions where we present directly for each example the path that the algorithm thinks is responsible for its selection, allowing the user to correct it. This allows to remove the restriction of bounding length of paths.

2. Regular Tree Queries

Another direction is also to extend to richer classes. Going from RPQ to conjunctions of RPQ requires to work on richer pattern than just paths. The more immediate extension is to use trees. This inner algorithm for the inference of regular tree languages can therefore be an algorithm such as *RPNI* for trees [GO93], instead of the word version.

The difficulty, however, remains in finding the tree pattern responsible for the selection of each example node. This problem, linked to the pruning problem in the tree case, can be rephrased the following way. Each example node is selected because there is a specific conjunction of paths that starts from it. The idea is to find the smallest conjunction of paths that would not allow the selection of a counter-example. Exactly how this can be done efficiently is still open to debate.

One solution we foresee for instance is to have an initial procedure that considers each example node, and that enumerates its paths, building a set of paths by adding each path one by one until the obtained pattern does not allow to select a counter-example. The resulting pattern would still contain too many paths, but another *cleaning* procedure can try to remove them one by one, as long as the remaining patterns does not allow the selection of a counter-example.

Consider for instance the query defined by the expression $\texttt{isFriendOf}^+$. An example node would contain many other outgoing edges, such `isChildOf` for instance. The first procedure enumerates all those edges until it captures `isFriendOf`. The reason would be that the expression `isChildOf` can also select some possible counter-example, while $\texttt{isChildOf} \wedge \texttt{isFriendOf}$ does not (but is still too restrictive). The cleaning procedure can remove the edge `isChildOf` from the pattern safely, this makes the query more general but without capturing any counter-examples.

This procedure would still have some problems. First, it ignores the fact that we have actually unordered trees. Second, it is probably not very efficient as it the search space seems important. It is still an open problem whether those problems can be overcome, and how.

3. Using Data Values

One of the major restrictions of our current approach on tree queries is the fact that the queries we infer are not able to exploit the data values presented in semi-structured data, focussing only on structural elements. Even if this covers a fair amount of queries, including some that a user would more naturally write using data values, it is nevertheless true that many *real-world* queries are using tests on data values.

Relations between data values also enrich structures as one of their most classical use is to establish implicit links between objects, by the use of foreign keys, and as argued before, this is probably one of the keys to define expressive queries on graphs.

Dealing with data values, and hence exploiting an infinite alphabet, in finite state models has been long studied. However, most fundamental results of finite state automata do not carry over to models using infinite alphabets. For instance, equivalence problem are undecidable for register automata and pebble automata [NSV04].

However, results of [BLP10] bring new hope to those problems. Deterministic Finite-Memory automata [KF94b] are essentially automata with registers that reads words with data values (each position of the word is associated with an element of an infinite alphabet). The automaton can, at any point, store a value in one of its register, or test the value of the current position with one of its registers. In [BLP10], some natural restrictions are added, such as the fact that two registers can not contain the same value. This allows to define a normal form, which in turn allow efficient equivalence tests. Those results can also serve as a base for machine learning algorithms, as indicated by [DHLT14].

We believe those results can be carried over even further to be integrated in our query inference approach. We foresee several possible extensions. First, we need to extend those results to **trees**. One way to do that is to extend bottom-up tree automata such that states hold registers. At each rule, the tree automata could compare the values of the current node, as well as values carried by each of its children. We believe this extension natural enough so that the result of word finite-memory machine can be carried to this model.

The expressiveness of the resulting class is not yet clear. However, if one consider **data tree queries** that can be derived from this object - using canonical encoding similar to the one we described in the section on queries - we can define queries that select pairs of nodes with the same values for instance. This allows to exploit implicit links defined by foreign keys, at least up to some points.

Finally, those techniques could also be applied to tree unfoldings of graph structures. In this case, the identifier of nodes can be stored as a data value,

and the equality relation between those identifiers can be used to express cyclic queries.

We think the study of this family of formal objects offer a viable cornerstone for the inference of tuple queries with data values. In particular the link between this model and MSO graph queries is certainly an important aspect to study.

5 Conclusion

In this document, I made a succinct presentation of more than fifteen years of research that mainly focused around the idea of adapting grammatical inference techniques to the problem of **query and transformation inference over structured data**. Our contributions in this field indeed allowed to obtain strong learnability results for interesting classes of queries on trees as well as classes of transformations.

Starting from existing grammatical inference algorithms such as OSTIA for word transducers [OGV93] or RPNI for word and tree automata [GO93], we designed learning algorithms for richer classes such as regular tree queries [CLN04, CGLN05, CGLN07, LNG06, CGLN08b], word rational functions [BLN12], regular tree-to-string transductions [LLN+11], top-down tree transductions [LMN10b, BLN16]. These algorithms haven't only improved learnability results, they also provided methods that can efficiently be used in practical scenarios of query and transformation inference.

This also allowed us to find results that also had impact to the larger field of formal language, and finite state machines. For instance, we managed to define normal forms and Myhill-Nerode theorems for several classes of transformations. This proved to be useful for equivalence checking for instance, or other static analysis problems.

For instance, one of our current fields of research, which is part of the ANR Project "Colis", consists in finding formalisms that can express a large family of Linux install scripts, and for which efficient procedures exist to perform analysis. This includes for instance verifying that the execution of an install script followed by the execution of its uninstall script let the global filesystem unchanged, or that two install scripts are dependant or not. This problem is not directly related to machine learning issues, but some similar techniques can be used to study this problem.

We can also cite works that we did during the thesis of A. Ndione [Ndi15, NLN13] on property testing for XML. This allowed us to design efficient tests if a large XML document has a good probability to comply a DTD schema or not with an amount of tests which does not depend directly on the size of the document.

This also allows to offer a real alternative to existing techniques which are mostly based on statistical approaches. Our approach is able to infer richer classes of queries and transformations in realistic scenarios. The usual limitation of the symbolic approaches - essentially a weakness toward noisy data - is non existent in our approach where our data are obtained with few - clean - interactions with an end-user.

Nevertheless, some questions remain open. For **tree query** inference, the main problem is that our technique does not yet make usage of **data values**. While in practice this limitation seems to be less dramatic than expected, it still remains an important limitation. However, as we argued

in a previous chapter, techniques based on data automata give some hope in this direction.

In the domain of **tree transformations**, the big question that remains is the question of **learnability of MSO transformations**. The class of Macro Tree Transducers with regular look-ahead provides a nice formalism, but there is no Myhill-Nerode theorem yet for this class which could serve as a basis for a learning algorithm. Both the question of inferring a transducer with a look-ahead, and to infer an MTTare still open. Furthermore, the difficulties encountered to capture the whole class of tree-to-string transducers - which can be seen as a subclass of MTT- let us think that it may be possible to capture only a subclass of MSO transformations, which is yet to be defined.

For **graph query** inference, results presented here are still part of preliminary works. The whole question of defining a class of queries expressive and learnable remains open, our ultimate goal being a class equivalent to **MSO definable queries on graphs**. On a more practical side, a learning plausible **scenario of interaction** with an end-user is also to define, with, among other problems, the question of visualisation of the graph database.

The inference of tree transformations and graph queries also opens the door to the inference of **graph transformations**. As graphs are one of the most general representation for data, in fact graph transformations cover a wide range of computer processing operations. In database however, this covers a more specific kind of applications.

For instance, the **data exchange** paradigm [FKMP05] aims at transforming data organised in a source schema to fit a target schema, which allows to import data from one database to another. In **data integration** [HAMA16], the data is not transformed directly, but a unified view over different sources is provided to the user. He can then define queries over the unified view, which are transformed to be expressed directly on any of the source formats. This allows to keep separated databases, with different schemas, but with strong implicit transformation links between them. Typically, this is done using **schema mapping** techniques. In this framework, the link between the source and the target schema is defined by a set of logical formulae.

Formally, this can be related to the graph transformation model of Courcelle [Cou97], in which a graph transformation is defined as a set of logical expressions. The nodes of the output graph are copies of the node of the input graph (possibly a bounded amount of copies), and each edge label corresponds to a predicate between those nodes, whose logical value (and hence the presence or not of the edge) is defined as a logical formula on the input graph. Technically, this simply means that graph transformations can be expressed as a set of graph queries.

The conception of graph transformations are even more complex for the user however than graph queries, as the whole output graph has to be defined. This makes inference techniques even more relevant. That is why we

propose to extend inference techniques for graph queries on graph transformations.

Nevertheless, the inference of graph transformations presents some inherent difficulties, both in a static and an interactive learning framework. Mostly the exact learning scenario has yet to be defined.

In a static framework, the learning algorithm would probably take as an input a graph (or a part of a graph) in the source schema and its corresponding counterpart in the target schema. This means however that a first process needs to align both graphs, using some kind of **entity alignment**. This problem is already studied, as in [BWCB11, NB12], but needs to be interfaced correctly with the learning algorithm. In particular, it is necessary for each edge of the output graph to define which part of the input graph is responsible for its creation, which includes possibly a whole subgraph. This problem, related to the problem of pruning trees or graphs, seems to be even more crucial in the problem of graph transformations as the different edges of the target graph have to be linked together.

The interactive framework may possibly allow to circumvent some of those difficulties by adding new interactions (that are yet to be defined), but it also presents again the problem of **data visualisation**. This problem is even harder than for graph queries as a whole output graph has to be presented to the end-user, and there is no apparent solution to do this in a nice intuitive way.

Graph transformations could also be applied on a single database, as it can also help to infer new data from the database itself. This means for instance that our approach could potentially have applications for instance in **ontology inference**, or **knowledge discovery**, through the inference of new concepts from the data. This could open the gate to new forms of database management tools, where the database can auto-evolve by adding new facts, that are induced from existing data, and from an analysis of the database.

Bibliography

[Abi97] Serge Abiteboul. Querying Semi-Structured Data. In **ICDT**, pages 1–18, 1997.

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. **Data on the Web: From Relational to Semistructured Data and XML**. Morgan Kaufmann Publishers Inc., 2000.

[ADK+15] Yael Amsterdamer, Susan B. Davidson, Anna Kukliansky, Tova Milo, Slava Novgorodov, and Amit Somech. Managing general and individual knowledge in crowd mining applications. In **CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings**, 2015.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. **Foundations of Databases**. Addison-Wesley Longman Publishing Co., Inc., 1995.

[AM09] Rajeev Alur and P. Madhusudan. Adding nesting structure to words. **Journal of the ACM**, 56(3):1–43, 2009.

[Ang87a] D. Angluin. Learning Regular Sets from Queries and Counterexamples. **Information and Computation**, 75(2):87–106, November 1987.

[Ang87b] Dana Angluin. Learning Regular Sets from Queries and Counterexamples. **Information and Computation**, 75(2):87–106, November 1987.

[Ang88] Dana Angluin. Queries and Concept Learning. **Machine Learning**, 2(4):319–342, April 1988.

[Bae03] Ricardo A. Baeza-Yates. Information retrieval in the web: beyond current search engines. **Int. J. Approx. Reasoning**, 34(2-3):97–104, 2003.

[Bag06] Guillaume Bagan. MSO Queries on Tree Decomposable Structures are Computable with Linear Delay. In **Computer Science Logic**, volume 4646 of **Lecture Notes in Computer Science**, pages 208–222. Springer Verlag, 2006.

[Bar03] Albert-László Barabási. **Linked - how everything is connected to everything else and what it means for business, science, and everyday life**. Plume, 2003.

[BCL15a] Angela Bonifati, Radu Ciucanu, and Aurélien Lemay. Interactive path query specification on graph databases. In **Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.**, pages 505–508, 2015.

[BCL15b] Angela Bonifati, Radu Ciucanu, and Aurélien Lemay. Learning path queries on graph databases. In **Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.**, pages 109–120, 2015.

[BG07] Irad Ben-Gal. Bayesian networks. **Encyclopedia of statistics in quality and reliability**, 2007.

[BH99] Marc Bernard and Colin De La Higuera. GIFT: Grammatical Inference for Terms. In **ILP**, 1999.

[BHLM13] Benedikt Bollig, Peter Habermehl, Martin Leucker, and Benjamin Monmege. A Fresh Approach to Learning Register Automata. In **Developments in Language Theory**, volume 7907 of **Lecture Notes in Computer Science**, pages 118–130, 2013.

[BHM04] Ricardo A. Baeza-Yates, Carlos A. Hurtado, and Marcelo Mendoza. Query recommendation using query logs in search engines. In **Current Trends in Database Technology - EDBT 2004 Workshops, EDBT 2004 Workshops PhD, DataX, PIM, P2P&DB, and ClustWeb, Heraklion, Crete, Greece, March 14-18, 2004, Revised Selected Papers**, pages 588–596, 2004.

[Bis95] C.M. Bishop. **Neural networks for pattern recognition**. Oxford University Press, USA, 1995.

[BK08] Michael Benedikt and Christoph Koch. XPath leashed. **ACM Computing Surveys**, 41(1):1–54, d 2008.

[BK09] Michael Benedikt and Christoph Koch. From XQuery to relational logics. **ACM Transactions on Database Systems**, 34(4), 2009.

[BKMW01] Anne Brüggemann-Klein, Makoto Murata, and Derick Wood. Regular Tree and Regular Hedge Languages over Unranked Alphabets: Version 1. Technical Report HKUST-TCSC-2001-05, April 2001.

[BL05] Pablo Barcelo and Leonid Libkin. Temporal Logics over Unranked Trees. In **20th Annual IEEE Symposium on Logic in Computer Science**, pages 31–40. IEEE Comp. Soc. Press, 2005.

[BLN12] Adrien Boiret, Aurélien Lemay, and Joachim Niehren. Learning Rational Functions. In Hsu C. Yen, Oscar H. Ibarra, Hsu C. Yen, and Oscar H. Ibarra, editors, **Developments in Language Theory**, volume 7410 of **Lecture Notes in Computer Science**, pages 273–283. Springer, 2012.

[BLN16] Adrien Boiret, Aurélien Lemay, and Joachim Niehren. Learning Top-Down Tree Transducers with Regular Domain Inspection. In **International Conference on Grammatical Inference 2016**, 2016.

[BLP10] M. Benedikt, C. Ley, and G. Puppis. What you must remember when processing data words. In **Proceedings of the 4th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW)**, volume 619 of **CEUR Workshop Proceedings**. CEUR-WS.org, 2010.

[BMS+06] Mikolaj Bojańczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-Variable Logic on Words with Data. In **21st Annual IEEE Symposium on Logic in Computer Science**, pages 7–16. IEEE Comp. Soc. Press, 2006.

[Boi16] Adrien Boiret. **Normalization and Learning of Transducers on Trees and Words. (Normalisation et Apprentissage de Transducteurs d'Arbres et de Mots)**. PhD thesis, Lille University of Science and Technology, France, 2016.

[Bor99] Andrew Eliot Borthwick. **A Maximum Entropy Approach to Named Entity Recognition**. PhD thesis, New York, NY, USA, 1999. AAI9945252.

[BP16] Adrien Boiret and Raphaela Palenta. Deciding equivalence of linear tree-to-word transducers in polynomial time. In **Developments in Language Theory - 20th International Conference, DLT 2016, Montréal, Canada, July 25-28, 2016, Proceedings**, pages 355–367, 2016.

[BWCB11] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In **Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence**, AAAI'11, pages 301–306. AAAI Press, 2011.

[Cal98] Mary Elaine Califf. Relational learning techniques for natural language information extraction. Technical report, 1998.

[Car05] Julien Carme. **Inférence de requêtes régulières dans les arbres et applications à l'extraction d'information sur le Web**. PhD thesis, Université Lille 3, 2005.

[CD99] James Clark and Steven DeRose. XML path language (XPath) version 1.0. W3C recommendation, W3C, November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[CDG+02] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques

and Applications. Available on: http://www.grappa.univ-lille3.fr/tata, 2002.

[CGLN05] Julien Carme, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Interactive Learning of Node Selecting Tree Transducers. In **IJCAI Workshop on Grammatical Inference**, 2005.

[CGLN07] Julien Carme, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Interactive Learning of Node Selecting Tree Transducers. **Machine Learning**, 66(1):33–67, January 2007.

[CGLN08a] Jérôme Champavère, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Efficient inclusion checking for deterministic tree automata and DTDs. In **2nd International Conference on Language and Automata Theory and Applications**, 2008.

[CGLN08b] Jérôme Champavère, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Schema-Guided Induction of Monadic Queries. In **9th International Colloquium on Grammatical Inference**, volume 5278 of **Lecture Notes in Computer Science**, pages 15–28. Springer Verlag, 2008.

[CGLN09] Jérôme Champavère, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Efficient inclusion checking for deterministic tree automata and XML schemas. **Information and Computation**, 207(11):1181–1208, 2009.

[CGLN10] Jérome Champavère, Rémi Gilleron, Aurélien Lemay, and Joachim Niehren. Schema-Guided Query Induction. 2010.

[CGLV02] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. **Journal of Computer and System Sciences**, 64(3):443 – 465, 2002.

[Cha10] Jérôme Champavère. **Induction de requêtes guidée par schéma**. PhD thesis, Université Lille 1, September 2010.

[Cho79] C. Choffrut. 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, 1979.

[Cho03] Christian Choffrut. Minimizing subsequential transducers: a survey. **Theoretical Computer Science**, 292(1):131–143, 2003.

[Ciu15] Radu Ciucanu. **Cross-Model Queries and Schemas: Complexity and Learning**. Theses, Université Lille 1, July 2015.

[Cla99] James Clark. XSL transformations (XSLT) version 1.0. W3C recommendation, W3C, November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.

[CLN04] Julien Carme, Aurélien Lemay, and Joachim Niehren. 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, 2004.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In **9th ACM Symposium on Theory of Computing**, pages 77–90. ACM-Press, 1977.

[CMW87a] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. **SIGMOD Rec.**, 16(3):323–330, December 1987.

[CMW87b] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. **SIGMOD Rec.**, 16(3):323–330, December 1987.

[CN02] Hai L. Chieu and Hwee T. Ng. A maximum entropy approach to information extraction from semi-structured and free text. In **Proceedings of Eighteenth national conference on Artificial intelligence**, pages 786–791, 2002.

[CNT04a] Julien Carme, Joachim Niehren, and Marc Tommasi. Querying Unranked Trees with Stepwise Tree Automata. In **19th International Conference on Rewriting Techniques and Applications**, volume 3091 of **Lecture Notes in Computer Science**, pages 105–118. Springer Verlag, 2004.

[CNT04b] Julien Carme, Joachim Niehren, and Marc Tommasi. Querying Unranked Trees with Stepwise Tree Automata. In **15-th International Conference on Rewriting Techniques and Applications**, 2004.

[Cou94] Bruno Courcelle. Monadic second-order definable graph transductions: a survey. **Theoretical Computer Science**, 126(1):53–75, 1994.

[Cou97] Bruno Courcelle. Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations. In Grzegorz Rozenberg, editor, **Handbook of Graph Grammars**, 1997.

[Cou09] Bruno Courcelle. Linear delay enumeration and monadic second-order logic. **Discrete Applied Mathematics**, 157(12):2675–2700, 2009.

[CRdR00] Boris Chidlovskii, Jon Ragetli, and Maarten de Rijke. Wrapper generation via grammar induction. In **Proceedings European Conference on Machine Learning (ECML'2000)**, number 1810 in LNCS. Springer, 2000.

[Cro09] Douglas Crockford. Nosql definition: : Next generation databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable. Technical report, 2009. http://nosql-database.org.

[Cro14] Douglas Crockford. Json specification. Technical Report RFC 7159, IETF, 2014. https://tools.ietf.org/html/rfc7159.

[css] Cascading style sheets, home page. `http://www.w3.org/Style/CSS/Overview.en.html`. W3C, Accessed: 2017-09-10.

[CSZ10] Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. **Semi-Supervised Learning**. The MIT Press, 1st edition, 2010.

[DHI12] AnHai Doan, Alon Halevy, and Zachary Ives. **Principles of Data Integration**. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2012.

[DHLT14] Normann Decker, Peter Habermehl, Martin Leucker, and Daniel Thoma. **Learning Transparent Data Automata**, pages 130–149. 2014.

[dlH97] C. de la Higuera. Characteristic Sets for Polynomial Grammatical Inference. **Machine Learning**, 27:125–137, 1997.

[dlHJ04] C. de la Higuera and J.C. Janodet. Inference of omega-languages from prefixes. **Theoretical Computer Science**, 313(2):295 – 312, 2004.

[EC12] Joost Engelfriet and Bruno Courcelle. **Graph Structure and Monadic Second-order Logic**. Number 138 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2012.

[Eil74] S. Eilenberg. **Automata, Languages and Machines**. Academic Press, 1974.

[ELL09] Brian S. Everitt, Sabine Landau, and Morven Leese. **Cluster Analysis**. Wiley Publishing, 4th edition, 2009.

[EM65] C. C. Elgot and G. Mezei. On relations defined by generalized finite automata. **IBM J. of Res. and Dev.**, 9:88–101, 1965.

[EM03] J. Engelfriet and S. Maneth. Macro tree translations of linear size increase are MSO definable. **SIAM J. Comput.**, 32:950–1006, 2003.

[EM05] J. Engelfriet and S. Maneth. The equivalence problem for deterministic MSO tree transducers is decidable. In R. Ramanujam and S. Sen, 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, 2005.

[EMS09] Joost Engelfriet, Sebastian Maneth, and Helmut Seidl. Deciding equivalence of top-down XML transformations in polynomial time. **Journal of Computer and System Science**, 75(5):271–286, 2009.

[Eng75] J. Engelfriet. Tree automata and tree grammars. 1975.

[Eng76] Joost Engelfriet. Top-down tree transducers with regular look-ahead. **Mathematical systems theory**, 10(1):289–303, 1976.

[Eng77] J. Engelfriet. Top-down tree transducers with regular look-ahead. **Math. Systems Theory**, 10:289–303, 1977.

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. **J. Comp. Syst. Sci.**, 31:71–146, 1985.

[FK00a] Dayne Freitag and Nicholas Kushmerick. Boosted wrapper induction. pages 577–583. AAAI Press, 2000.

[FK00b] Dayne Freitag and Nicholas Kushmerick. Boosted Wrapper Induction. In **17-th National Conference on Artificial Intelligence**, 2000.

[FKMP05] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. **Theoretical Computer Science**, 336(1):89 – 124, 2005. Database Theory.

[FM99] Dayne Freitag and Andrew K. McCallum. Information extraction with HMMs and shrinkage. In **AAAI Workshop on Machine Learning for Information Extraction**, 1999.

[Fre00] Dayne Freitag. Machine learning for information extraction in informal domains. **Machine Learning**, 39(2):169–202, 2000.

[FRR+10] Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. Properties of Visibly Pushdown Transducers. In **35th International Symposium on Mathematical Foundations of Computer Science (MFCS'10)**, volume 6281 of **Lecture Notes in Computer Science**, pages 355–367. Springer Verlag, 2010.

[FSM10] Sylvia Friese, Helmut Seidl, and Sebastian Maneth. Minimization of Deterministic Bottom-Up Tree Transducers. In

Yuan Gao, Hanlin Lu, Shinnosuke Seki, and Sheng Yu, editors, **Developments in Language Theory, 14th International Conference DLT 2010**, volume 6224 of **Lecture Notes in Computer Science**, pages 185–196. Springer Verlag, 2010.

[GK04] Georg Gottlob and Christoph Koch. Monadic Datalog and the expressive power of languages for Web information extraction. **Journal of the ACM**, 51(1):74–113, January 2004.

[GKP05] Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing XPath queries. **ACM Transactions on Database Systems**, 30(2):444–491, 2005.

[GMM03] Ramanathan V. Guha, Rob McCool, and Eric Miller. Semantic search. In **Proceedings of the Twelfth International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20-24, 2003**, pages 700–709, 2003.

[GMTT06] Rémi Gilleron, Patrick Marty, Marc Tommasi, and Fabien Torre. Interactive Tuples Extraction from Semi-Structured Data. In **2006 IEEE/WIC/ACM International Conference on Web Intelligence**, 2006.

[GO93] Pedro García and Jose Oncina. Inference of Recognizable Tree Sets. Technical Report DSIC-II/47/93, Universidad de Alicante, 1993.

[Gol78] E. M. Gold. Complexity of Automaton Identification from Given Data. **Inform. Control**, 37:302–320, 1978.

[HAMA16] Mohammad Haghighat, Mohamed Abdel-Mottaleb, and Wadee Alhalabi. Discriminant correlation analysis: Real-time feature level fusion for multimodal biometric recognition. **Trans. Info. For. Sec.**, 11(9):1984–1996, September 2016.

[Hic12] Ian Hickson. Html5 specification. Technical Report 1.5610, World Wide Web Consortium, 2012. https://www.w3.org/TR/html5/.

[IK02] Hideki Isozaki and Hideto Kazawa. Efficient support vector classifiers for named entity recognition. In **Proceedings of the 19th International Conference on Computational Linguistics - Volume 1**, COLING '02, pages 1–7, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[Jac08] Matthew O. Jackson. **Social and Economic Networks**. Princeton University Press, Princeton, NJ, USA, 2008.

[JFY09] T. Joachims, T. Finley, and Chun-Nam Yu. Cutting-plane training of structural svms. **Machine Learning**, 77(1):27–59, 2009.

[KBBdB06] Raymond Kosala, Hendrik Blockeel, Maurice Bruynooghe, and Jan Van den Bussche. Information extraction from structured documents using **k**-testable tree automaton inference. **Data Knowl. Eng.**, 58(2):129–158, 2006.

[KF94a] Michael Kaminski and Nissim Francez. Finite-memory automata. **Theor. Comput. Sci.**, 134(2):329–363, November 1994.

[KF94b] Michael Kaminski and Nissim Francez. Finite-memory automata. **Theor. Comput. Sci.**, 134(2):329–363, 1994.

[KP94] Juhani Karhumäki and Wojciech Plandowski. On the Size of Independent Systems of Equations in Semigroups. In Igor Pr'ıvara, Branislav Rovan, Peter Ruzicka, Igor Pr'ıvara, Branislav Rovan, and Peter Ruzicka, editors, **MFCS**, volume 841 of **Lecture Notes in Computer Science**, pages 443–452. Springer, 1994.

[Kus97] N. Kushmerick. **Wrapper Induction for Information Extraction**. PhD thesis, University of Washington, 1997.

[KVBB02] Raymond Kosala, Jan Van Den Bussche, Maurice Bruynooghe, and Hendrik Blockeel. Information Extraction in Structured Documents using Tree Automata Induction. In **6th International Conference Principles of Data Mining and Knowledge Discovery**, pages 299–310, 2002.

[Lan92] K. J. Lang. Random DFA's can be approximately learned from sparse uniform examples. In **Proc. 5th Annu. Workshop on Comput. Learning Theory**, pages 45–52. ACM Press, New York, NY, 1992.

[Lau14] Grègoire Laurence. **Normalisation et Apprentissage de Transductions dâĂŹArbres en Mots**. PhD thesis, Université Lille 1, 2014.

[Lib06] Leonid Libkin. Logics over unranked trees: an overview. **Logical Methods in Computer Science**, 3(2):1–31, 2006.

[LIJ+15] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia. **Semantic Web**, 6(2):167–195, 2015.

[LLN+11] Grégoire Laurence, Aurélien Lemay, Joachim Niehren, Slawek Staworko, and Marc Tommasi. Normalization of Sequential Top-Down Tree-to-Word Transducers. In Adrian H. Dediu, Shunsuke Inenaga, Carlos M. Vide, Adrian H. Dediu, Shunsuke

Inenaga, and Carlos M. Vide, editors, **LATA**, volume 6638 of **Lecture Notes in Computer Science**, pages 354–365. Springer, 2011.

[LLN+14] Grégoire Laurence, Aurélien Lemay, Joachim Niehren, Slawek Staworko, and Marc Tommasi. Learning Sequential Tree-to-Word Transducers. In **8th International Conference on Language and Automata Theory and Applications**, Madrid, Espagne, March 2014. Springer.

[LMN10a] Aurélien Lemay, Sebastian Maneth, and Joachim Niehren. A Learning Algorithm for Top-Down XML Transformations. In **29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems**, pages 285–296. ACM-Press, 2010.

[LMN10b] Aurélien Lemay, Sebastian Maneth, and Joachim Niehren. 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, 2010. ACM Press.

[LMP01] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In **Proceedings of the Eighteenth International Conference on Machine Learning**, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[LNG06] Aurélien Lemay, Joachim Niehren, and Rémi Gilleron. Learning n-Ary Node Selecting Tree Transducers from Completely Annotated Examples. In **8-th International Colloquium on Grammatical Inference**, 2006.

[Mah06] Michael Mahemoff. **Ajax Design Patterns**. O'Reilly Media, Inc, 2006.

[Mar05] Maarten Marx. Conditional XPath. **ACM Transactions on Database Systems**, 30(4):929–959, d 2005.

[Mar07] Patrick Marty. **Induction d'extraction n-aire pour les documents semi-structurés**. PhD thesis, Université Charles de Gaulle, Lille 3, 2007.

[Md05] Maarten Marx and Maarten de Rijke. Semantic characterizations of navigational XPath. **ACM SIGMOD Record**, 34(2):41–46, June 2005.

[Mea55] George H. Mealy. A method for synthesizing sequential circuits. **Bell System Technical Journal**, 34(5):1045–1079, 1955.

[MMK99] Ion Muslea, Steven Minton, and Craig A. Knoblock. A hierarchical approach to wrapper induction. In **Agents**, pages 190–197, 1999.

[MMP03] James Mayfield, Paul McNamee, and Christine Piatko. Named entity recognition using hundreds of thousands of features. In **Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4**, CONLL '03, pages 184–187, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.

[MMP13] Amaldev Manuel, Anca Muscholl, and Gabriele Puppis. Walking on data words. In **Computer Science - Theory and Applications - 8th International Computer Science Symposium in Russia, CSR 2013, Ekaterinburg, Russia, June 25-29, 2013. Proceedings**, pages 64–75, 2013.

[Moo56] Edward F. Moore. Gedanken Experiments on Sequential Machines. In **Automata Studies**, pages 129–153. Princeton U., 1956.

[MPSM+04] Eve Maler, Jean Paoli, C. M. Sperberg-McQueen, François Yergeau, and Tim Bray. Extensible markup language (XML) 1.0 (third edition). Technical report, W3C, February 2004. http://www.w3.org/TR/2004/REC-xml-20040204.

[Mur00] M. Murata. Hedge Automata: a Formal Model for XML Schemata. Web page, 2000.

[Myh57] J. Myhill. Finite Automata and the Representation of Events. Technical Report 57-624, WADC, 1957.

[NB12] Duy Hoa Ngo and Zohra Bellahsene. YAM++ : (not) Yet Another Matcher for Ontology Matching Task. In Nicolas Anciaux, editor, **BDA: Bases de Données Avancées**, Clermont-Ferrand, France, October 2012.

[NCGL12] Joachim Niehren, Jérôme Champavère, Rémi Gilleron, and Aurélien Lemay. Query Induction with Schema-Guided Pruning Strategies, July 2012.

[NCGL13] Joachim Niehren, Jérôme Champavère, Rémi Gilleron, and Aurélien Lemay. Query Induction with Schema-Guided Pruning Strategies. **Journal of Machine Learning Research**, 2013.

[Ndi15] Antoine Mbaye Ndione. **Approximate membership for words and trees**. PhD thesis, Université Lille 1, 2015.

[Ner58] A. Nerode. Linear Automaton Transformation. In **Proc. American Mathematical Society**, volume 9, pages 541–544, 1958.

[NLN13] Antoine Ndione, Aurélien Lemay, and Joachim Niehren. Approximate membership for regular languages modulo the edit distance. **Theor. Comput. Sci.**, 487:37–49, 2013.

[NPTT05] Joachim Niehren, Laurent Planque, Jean-Marc Talbot, and Sophie Tison. N-ary queries by tree automata. In **Foundations of Semistructured Data, 6.-11. February 2005**, 2005.

[NSV04] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. **ACM Trans. Comput. Logic**, 5(3):403–435, July 2004.

[OG92] J. Oncina and P. Garcia. Inferring regular languages in polynomial update time. In **Pattern Recognition and Image Analysis**, pages 49–61, 1992.

[OG93] J. Oncina and P. García. Inference of recognizable tree sets. Technical report, Departamento de Sistemas Informáticos y Computación, Universidad de Alicante, 1993.

[OGV93] J. Oncina, P. Garcia, and E. Vidal. Learning Subsequential Transducers for Pattern Recognition and Interpretation Tasks. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, 15:448–458, 1993.

[OV96] J. Oncina and M. A. Varo. 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, 1996.

[Pot94] Andreas Potthoff. Modulo-counting quantifiers over finite trees. **Theoretical Computer Science**, 126(1):97 – 112, 1994.

[PPP+12] Hyunjung Park, Richard Pang, Aditya G. Parameswaran, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: A system for declarative crowdsourcing. **PVLDB**, 5(12):1990–1993, 2012.

[Qui87] J. R. Quinlan. Simplifying decision trees. **Int. J. Man-Mach. Stud.**, 27(3):221–234, September 1987.

[Rab69] M. O. Rabin. Decidability of Second-Order Theories and Automata on Infinite Trees. **Transactions of the American Mathematical Society**, 141:1–35, 1969.

[RBVdB08] Stefan Raeymaekers, Maurice Bruynooghe, and Jan Van den Bussche. Learning (k,l)-Contextual Tree Languages for Information Extraction from Web Pages. **Machine Learning**, 71(2-3):155–183, June 2008.

[rdf] Resource description framework. `http://www.w3.org/RDF/`. W3C, Accessed: 2017-09-10.

[rel] Relax ng home page. `http://relaxng.org/`. Accessed: 2017-09-10.

[RJBS08] Ganesh Ramakrishnan, Sachindra Joshi, Sreeram Balakrishnan, and Ashwin Srinivasan. **Using ILP to Construct Features for Information Extraction from Semi-structured Text**, pages 211–224. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Rou70] W. C. Rounds. Mappings and grammars on trees. **Math. Systems Theory**, 4:257–287, 1970.

[RS91] C. Reutenauer and M. P. Schützenberger. Minimalization of rational word functions. **SIAM Journal on Computing**, 20:669–685, 1991.

[RS08] Jean-François Raskin and Frédéric Servais. Visibly Pushdown Transducers. In **Automata, Languages and Programming, 35th International Colloquium**, volume 5126 of **Lecture Notes in Computer Science**, pages 386–397. Springer Verlag, 2008.

[RSC+10] Jonathan Robie, Jerome Simeon, Don Chamberlin, Daniela Florescu, Mary Fernandez, and Scott Boag. XQuery 1.0: An XML query language (second edition). W3C recommendation, W3C, December 2010. http://www.w3.org/TR/2010/REC-xquery-20101214/.

[Sch77] Marcel Paul Schützenberger. Sur une variante des fonctions sequentielles. **Theor. Comput. Sci.**, 4(1):47–57, 1977.

[SF13] Pramod J. Sadalage and Martin Fowler. **NoSQL distilled : a brief guide to the emerging world of polyglot persistence**. Addison-Wesley, Upper Saddle River, NJ, 2013.

[SLLN09] Slawek Staworko, Grégoire Laurence, Aurélien Lemay, and Joachim Niehren. 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, 2009.

[Sod99a] Stephen Soderland. Learning Information Extraction Rules for Semi-Structured and Free Text. **Machine Learning**, 34(1-3):233–272, 1999.

[Sod99b] Stephen Soderland. Learning information extraction rules for semi-structured and free text. **Machine Learning**, 34(1):233–272, 1999.

[spa] Sparql 1.1 overview. https://www.w3.org/TR/2013/REC-sparql11-overview-20130321. W3C Recommendation 21 March 2013, Accessed: 2017-09-10.

[SS01] Bernhard Scholkopf and Alexander J. Smola. **Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond**. MIT Press, Cambridge, MA, USA, 2001.

[tCFL10] Balder ten Cate, Gaëlle Fontaine, and Tadeusz Litak. Some modal aspects of xpath. **Journal of Applied Non-Classical Logics**, 20(3):139–171, 2010.

[tCM07] Balder ten Cate and Maarten Marx. Axiomatizing the logical core of XPath 2.0. In **International Conference on Database Theory**, 2007.

[THJA04] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In **International Conference on Machine Learning (ICML)**, pages 104–112, 2004.

[Tho90] W. Thomas. **Handbook of Theoretical Computer Science**, volume B, chapter Automata on Infinite Objects, pages 134–191. MIT Press, 1990.

[Tra50] B. A. Trakhtenbrot. Impossibility of an algorithm for the decision problem in finite classes. **Doklady Akademii Nauk SSSR**, 70:569–572, 1950.

[TW68] J. W. Thatcher and J. B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. **Mathematical System Theory**, 2:57–82, 1968.

[xml] Xml schema part 1: Structures (second edition). http://www.w3.org/standards/techs/xmlschema. W3C Recommendation, Accessed: 2017-09-10.

[xpaa] Xml path language (xpath) 2.0 (second edition). http://www.w3.org/TR/xpath20/. W3C Recomendation, Accessed: 2017-09-10.

[xpab] Xml path language (xpath) 3.0. http://www.w3.org/TR/xpath-30/. W3C Recomendation, Accessed: 2017-09-10.

[Yan81] Mihalis Yannakakis. Algorithms for Acyclic Database Schemes. In **Proceeding of VLDB**, pages 82–94, 1981.