UNIVERSIDADE TÉCNICA DE LISBOA
INSTITUTO SUPERIOR TÉCNICO

Using Grammar Inference Techniques in Ontology Learning

André Luís Martins
(Licenciado)

Dissertação para Obtenção do Grau de
Mestre em Engenharia Informática e de Computadores

Documento Provisório

Junho 2006

Abstract

Documents in general, and technical papers in particular, share an implicit structure. This work aims at the development of an ontology learning method that allows the formalization of the structure of technical papers in a semi-automatic way. This should enable more elaborate searches to be performed on them, as well as improve precision and recall values of existing searches by allowing them to focus on specific parts of the documents.

The work is divided into three parts: the first part deals with the division and identification of a set of basic concepts in the source documents; the second part deals with methods for learning a compact description of the implicit document structure using the basic concepts identified in the first part as tokens; finally, the last part aims at transforming this compact representation into a formal ontology.

An approach applying a Naive Bayes classifier is proposed for the identification of the basic concepts, after segmenting the source documents, based on their formating, using a special purpose parser. The source documents used in this work are written in HTML, allowing easy access to the formating information.

To learn a compact description of the implicit document structure, two approaches were developed. The first is based on a modification of the active learning algorithm L*, while the second is based on the application of the Minimum Description Length principle to perform Deterministic Finite Automata inference.

The transformation from a compact description into a formal ontology is made by recursive identification of local patterns, that are captured as new ontology concepts.

As a byproduct, the ontology learning method also enables the use of the information obtained during the building process to extract concept instances from the source documents, thereby populating the ontology or allowing its annotation.

Keywords

Ontology Learning; Language Inference; Regular Sets; Document Structure Inference.

Resumo

Documentos em geral, e artigos científicos em particular, partilham uma estrutura implícita. Neste trabalho pretende-se desenvolver um método de aprendizagem de ontologias que permita a formalização da estrutura de artigos científicos de forma semi-automática. Isto permitirá realizar pesquisas mais avançadas sobre os mesmos, para além de melhorar a precisão ("precision") e cobertura ("recall") dos resultados de pesquisas normais ao permitir que estas se foquem em partes específicas dos documentos.

O trabalho está dividido em três partes: a primeira delas lida com a divisão e identificação de um conjunto de conceitos-base nos documentos-fonte; a segunda parte lida com a aprendizagem de uma descrição compacta da estrutura implícita dos documentos usando os conceitos-base identificados na primeira parte; finalmente, a última parte procura obter uma transformação desta representação compacta para uma ontologia formal.

É proposta uma abordagem que aplica um classificador do tipo "Naive Bayes" para a identificação dos conceitos-base, depois de aplicado um algoritmo especializado para a segmentação dos documentos-fonte com base na sua formatação. Os documentos-fonte usados neste trabalho estão escritos em HTML, permitindo fácil acesso à sua formatação.

Para aprender uma descrição compacta da estrutura implícita dos documentos, foram desenvolvidas duas abordagens. A primeira é baseada numa modificação do algoritmo de aprendizagem activa L*, e a segunda é baseada na aplicação do princípio da descrição mais compacta à inferência de autómatos finitos deterministas.

A transformação de uma descrição compacta para uma ontologia formal é feita através da identificação recursiva de padrões locais, sendo estes capturados como novos conceitos na ontologia.

Como resultado secundário do método de aprendizagem de ontologias, este permite o uso da informação obtida durante a construção da ontologia para extrair instâncias de conceitos, dos documentos fonte, para "povoar" a ontologia ou permitir a sua anotação.

Palavras-chave

Aprendizagem de ontologias; Inferência de linguagens; Conjuntos Regulares; Inferência da Estrutura de Documentos.

Resumo Alargado

Na internet existem biliões de documentos sobre os mais variados assuntos, prontos para serem encontrados por motores de pesquisa. Os motores de pesquisa actuais usam várias técnicas, mas todas elas são essencialmente baseadas na pesquisa por palavras chave. Esta forma de pesquisa resulta normalmente numa grande quantidade de documentos, muitos dos quais pouca ou nenhuma relevância têm. Como tal, seria interessante reduzir o número de documentos obtidos, usando conhecimento sobre o significado da informação presente nos documentos – a semântica.

Um dos objectivos da "Semantic Web" é o uso da semântica da informação que se encontra na internet para permitir pesquisas mais sofisticadas e precisas.

As ontologias providenciam uma definição formal de um domínio que pode ser partilhada entre pessoas e sistemas e aplicações heterogéneas. As ontologias são normalmente compostas por um conjunto de termos ou conceitos, relações entre estes e axiomas, limitando a sua expressão e desta forma restringindo a sua interpretação, ou seja, definindo a sua semântica. Os axiomas são expressos numa lógica formal. Assim, as ontologias são o candidato ideal para expressar a semântica que se pretende dar à informação contida nos documentos. Para tal é necessário que os documentos estejam anotados com essa informação e que existam motores de pesquisa especializados que tirem partido dessa informação. Logo, precisamos tanto de ontologias como de informação anotada segundo essas ontologias.

O processo de construção de ontologias a partir do zero é moroso, sendo um processo normalmente manual, o qual exige engenheiros de ontologias altamente especializados. Actualmente a construção de ontologias é mais uma arte do que uma tarefa de engenharia. Para além disso, a anotação manual de um grande número de documentos segundo a ontologia seria um processo extremamente moroso, senão mesmo impraticável. Assim torna-se útil a construção de um método automático ou semi-automático que realize a construção da ontologia e permita uma fácil anotação dos documentos.

A tarefa da construção automática de ontologias é o objectivo da área de aprendizagem de ontologias, que junta conhecimento das áreas de Aprendizagem ("Machine Learning"), Processamento de Língua Natural ("Natural Language Processing"), entre outras.

Documentos em geral, e artigos científicos em particular, partilham uma estrutura implícita. Neste trabalho pretende-se desenvolver um método de aprendizagem de ontologias que permita a formalização da estrutura de artigos científicos de forma semi-automática. Isto permitirá realizar pesquisas mais avançadas sobre os mesmos, para além de melhorar a precisão ("precision") e cobertura ("recall") dos resultados de pesquisas normais ao permitir que estas se foquem em partes específicas dos documentos.

O trabalho está dividido em três partes: a primeira delas lida com a divisão e identi-

ficação de um conjunto de conceitos-base nos documentos-fonte; a segunda parte lida com a aprendizagem de uma descrição compacta da estrutura implícita dos documentos usando os conceitos-base identificados na primeira parte; finalmente, a última parte procura obter uma transformação desta representação compacta para uma ontologia formal.

É proposta uma abordagem que aplica um classificador do tipo "Naive Bayes" para a identificação dos conceitos-base, depois de aplicado um algoritmo especializado para a segmentação dos documentos-fonte com base na sua formatação. A segmentação depende de um conjunto de atributos, os quais representam características do texto e da sua formatação. A segmentação dos documentos-fonte é feita com base na variação dos atributos baseados na formatação do texto. Os documentos-fonte usados neste trabalho estão escritos em HTML, permitindo fácil acesso à sua formatação. Com a abordagem proposta foram obtidos resultados promissores, apresentando de forma geral um baixo erro de classificação. Alguns casos particulares obtiveram no entanto taxas de erro maiores, isto porque a sua identificação é particularmente difícil devido à grande variedade de formatações que são usadas para os apresentar.

A descrição compacta escolhida foi um autómato finito determinista. Para aprender uma descrição compacta da estrutura implícita dos documentos, foram desenvolvidas duas abordagens.

1. A primeira é baseada numa modificação do algoritmo de aprendizagem activa L*. Este algoritmo usa um modelo de inferência em que o algoritmo pode fazer perguntas a um "professor" ou "oráculo" por forma a tentar aprender o que se pretende. Nesta abordagem pretendeu-se que o "oráculo" seja uma pessoa que iria responder às perguntas do algoritmo, no entanto o algoritmo L* coloca um número excessivo de perguntas. Por forma a reduzir o número de perguntas necessárias para a inferência, tendo em vista a interacção com uma pessoa, é proposta uma modificação do algoritmo que permite ao utilizador indicar uma justificação para a resposta negativa a algumas perguntas. Esta informação extra é utilizada para a redução do número de perguntas. Com esta modificação, obtiveram-se reduções muito significativas no número de perguntas necessárias para inferência.

2. A segunda abordagem é baseada na aplicação do princípio da descrição mais compacta à inferência de autómatos finitos deterministas. Com esta abordagem pretende-se um método de inferência que utilize apenas exemplos positivos (documentos de um mesmo tipo, para o qual se pretende descobrir a estrutura) e dispense intervenção humana, tornando desta forma o processo mais automático. Esta abordagem utiliza o princípio da descrição mais compacta como forma de avaliar cada hipótese, no contexto de um algoritmo de procura. Foi obtido sucesso na inferência de um

conjunto de pequenos autómatos sintéticos, mas a abordagem apresenta um problema de escalabilidade devido ao espaço de procura crescer muito com o tamanho do conjunto de exemplos fornecido ao algoritmo.

A transformação de uma descrição compacta para uma ontologia formal é feita através da identificação recursiva de padrões locais, sendo estes capturados como novos conceitos na ontologia. A transformação é guiada por uma heurística, com a qual se pretende controlar a "qualidade" ou aspecto da ontologia final. Esta abordagem obteve bons resultados, no entanto não é directamente aplicável à classe completa dos autómatos finitos deterministas. Pensa-se que esta dificuldade não será um problema para a aplicação em questão.

Como resultado secundário do método de aprendizagem de ontologias, este permite o uso da informação obtida durante a construção da ontologia para extrair instâncias de conceitos, dos documentos fonte, para "povoar" a ontologia ou permitir a sua anotação.

Acknowledgments

I would like to thank my supervisors, professor Helena Sofia Pinto and professor Arlindo Oliveira, for without their patient and continuous support, this work would surely not have reached a fruitful end.

I wish as well to thank my colleagues and friends at INESC-ID for the great work environment they helped to create and for many fruitful idea discussions.

I also benefited from an enlightening discussion with Professor Pieter Adriaans, from the University of Amsterdam, whom I would like to thank, both for the discussion and for accepting to participate in the jury of this thesis.

This work was partially supported by "Fundação para a Ciência e Tecnologia" with a research grant under the PSOC/EIA/58210/2004 (OntoSeaWeb-Ontology Engineering for the Semantic Web) project.

Last but not the least, I also wish to leave a word of strong affection to my parents and my sister for their continuous encouragement and motivation.

Contents

List of Abbreviations

DC Dublin Core Metadata

DFA Deterministic Finite Automaton

DOM Document Object Model

DTD Document Type Definition

FOL First Order Logic

FSA Finite State Automaton

HTML HyperText Markup Language

KIF Knowledge Interchange Format

MAP Maximum *a posteriori*

MDL Minimum Description Length

NFA Non-Deterministic Finite Automaton

NLP Natural Language Processing

OWL Web Ontology Language

PDF Portable Document Format

POS Part-of-Speech

PTA Prefix Tree Automaton

RDF Resource Description Framework

rDFA Reduced Deterministic Finite Automaton

SOTA Self Organizing Tree Algorithm

TFIDF Term Frequency-Inverted Document Frequency

UML Unified Modeling Language

XHTML eXtensible HyperText Markup Language

XML eXtensible Markup Language

List of Figures

List of Tables

Chapter 1

Introduction

There exist today millions of documents, on the Internet, about various topics, ready to be found by search engines. Current search engines use several techniques, but they are all essentially keyword based. This method either results in large amounts of documents, many of which are of little or no relevance, or in no results at all. Setting aside active search engine misdirection by the page authors, bad search results come either from a discrepancy between what is written in the pages and the keywords used in the search, or derive from a discrepancy between the context or word sense of the query and that of the web pages returned by the search. As such, it would be interesting to reduce the number of documents obtained, using knowledge about the meaning of the information contained in them – i.e. semantics. This is also one of the goals of the Semantic Web [1]: to add a well defined and machine processable meaning to the information available online. With this information available for machine processing, not only search engines can benefit from it, allowing for more elaborate searches, but also automated agents can communicate and offer services online without the need to have a fixed predefined interaction programmed into their code.

Ontologies provide a conceptualization of a domain that can be shared among people, computer systems and applications. They are usually composed of sets of concepts with relations among them and axioms involving these concepts and relations. The relations and axioms restrict the expression of concepts, therefore restricting their semantic interpretation. Sometimes a distinction is made between concepts and the terms associated with them.

Ontologies can have different degrees of formalization, ranging from informal categorizations such as library catalogs to ontologies written in formal languages with general logic constraints and axioms. For ontologies to be useful for computer processing, they must be written in a formal language.

Formal ontologies allow for the representation of the semantic of domain concepts

and, as such, are the ideal candidate for expressing the semantics we wish to add to the information contained in the documents found on the web. To enable the Semantic Web vision, still requires, among other things, to create these ontologies and to provide some means of association between them and the concept instances or occurrences in documents. This process of association is usually called annotation, whereby concept instances and relations in a document are associated with the concepts and relations in a given ontology by some form of embedding this information in the document.

To build ontologies from scratch is a costly and time consuming task, being a manual process that requires highly specialized ontology engineers. Currently, ontology construction is more of an art than an engineering task. Furthermore, manual annotation of a large number of papers, according to an ontology, would be an extremely time consuming task, therefore unpractical. So, it is useful to develop an automatic or semi-automatic process for ontology construction that should allow for an easy annotation of documents.

The automatic ontology construction task is the goal of the ontology learning research area, which brings together knowledge from Machine Learning, Natural Language Processing, among other areas. Ontology learning methods can be grouped according to their source type: text, dictionaries, relational schemas, semi-structured data. A particular domain of application for ontology learning techniques is that of inference of document structure. Documents can be organized into groups according to their implicit internal structure. For example, technical papers have a recognizable structure with parts such as title, sections, etc. It would be interesting to develop a method to infer an ontology that would describe such structure, allowing this semantic information to be used in the elaboration of advanced search queries. For example, one could search for papers by a particular author that mention some relevant work. This requires the identification of the author's name to be restricted to the beginning of the document so as not to mistakenly find it in the reference section and also to search for the desired work in the body of the document.

1.1 Goals and Contributions

The main goal of this work is the development of a semi-automatic ontology learning method for the structure of technical papers in HTML that would enable (1) the annotation or transformation of the papers into another format, like a database, and (2) to elaborate searches about them.

The method developed is divided into three parts:

1. Tokenization of the source documents: a method for tokenization and classification based a Naive Bayes classifier and a feature extraction algorithm for HTML

documents was developed [2].

2. Inference of a description of the implicit structure of documents: two different contributions were made in this part, the first was an improvement to the L* algorithm to reduce the number of queries with a human teacher [3], and the second was an inference method for Deterministic Finite Automata using the Minimum Description Length principle.

3. Transformation of that description into an ontology: an heuristic guided method to transform Deterministic Finite Automata into ontologies was developed.

1.2 Layout of the Thesis

In Chapter 2, a brief review of related work is presented. Since the thesis is focused on ontology learning methods, the chapter starts with a description of what is an ontology and what is required to build it, in Section 2.1, followed by a review of ontology learning methods, in Section 2.2. Work on wrapper induction, especially from HTML pages, has some similarities to the approach of this thesis, and is briefly overviewed in Section 2.3. Section 2.4 contains an overview of regular language learning, as this is important to a significant part of the learning method.

The learning method developed in this thesis is presented in Chapter 3. It is split into three main sections. The first, Section 3.1, addresses the source document tokenization problem using a Naive Bayes like classifier. In Section 3.2, two regular language learning approaches are used to capture the structure of the source documents: an active learning approach based on the L* algorithm, in Section 3.2.1, and an inference method based on the Minimum Description Length principle, in Section 3.2.2. In Section 3.3, an approach is proposed to complete the process, transforming the language, as captured by a Deterministic Finite Automaton, into an ontology. Also addressed is the problem of extracting ontology instances re-using the information obtained from the learning process.

Finally, Chapter 4 contains a brief discussion of the method developed, its strengths and weaknesses, along with conclusions and possible extensions of this work.

Chapter 2

Related Work

2.1 Ontology

To communicate information, we need a medium (writing, speech), a syntax (natural language grammar) and shared semantic symbols (words, verbs, etc.). This holds for people and, also, for computer systems, but with more strict and formal syntax for the latter. The shared semantics corresponds to a model of reality, that is at least partially agreed upon by those who use it to communicate.

Ontologies, having their origin in philosophy, address the last element, by providing a set of shared semantic symbols. They try to provide a shared view of the world, or of a specific domain, in an attempt to support formal reasoning about it.

Its first uses in computer systems were prompted by the need to share knowledge between large knowledge bases when building expert systems in Artificial Intelligence. For example, knowledge bases about diseases, their causes and symptoms, were used to build expert systems to perform automatic diagnosis, such as MYCIN. In another application, knowledge bases about chemical elements were used in expert systems to detect them. Many other applications in expert systems have been developed. These large knowledge bases were usually built from scratch and were very costly. Ontologies were proposed as a way to describe the common knowledge represented in these knowledge bases and allow sharing among them in order to reduce construction costs.

As mentioned above, not all ontologies have the same abstraction level. The most common ontologies are about a specific domain and were usually developed with a particular application in mind (such as the development of medical systems, like the Unified Medical Language System [4]). Others capture more general theories, such as the theory of time intervals [5], that are of interest in many domains and not only help reduce the cost of building ontologies but also help to reduce gratuitous divergences among them. Still others, more in line with the original philosophic ideal, attempt to capture every-

thing, at least in a general fashion. These are called "upper ontologies" [6, 7] and usually provide a top-level taxonomy in which to place all other concepts. Their use helps reduce divergence and eases matching of concepts between different ontologies by means of the concepts common base.

Some consider the term ontology to apply to a more broad spectrum than just formal ontologies (ontologies encoded in a formal language), encompassing things like Thesauri and term glossaries. However, in order to be of use to a computer system, the ontology must be encoded in a formal language. There are several formal languages available, like First Order Logic (FOL), Knowledge Interchange Format (KIF) [8] and OWL [9], among others. Each has different expressive power. An important issue in choosing a formal language is the tradeoff between its expressive power and the ability to build efficient inference engines supporting that expressiveness.

As a result of the formal language used to encode ontologies, their structure can vary significantly. There is, however, a basic structure commonly used, formed by the following elements:

- Concepts
- Relations
- Axioms

Concepts are the basic element of an ontology as they will be the object of inference in computer systems. Their definition, in formal terms, is given by the relations in which they participate and by the axioms that involve them. These restrict the possible meanings of concepts, distinguishing them from each other in a useful way for computer systems.

In some cases there is a separation between a concept (or relation) and the name that is associated with it. This separation is useful in ontologies that deal with natural language, where words can have multiple meanings and so be associated with different concepts (or relations), and there can be more than one word to describe the same concept (or relation). Another use is in the translation of the ontology into several languages when the ontology is to be used by people from different countries.

Most relations present in ontologies are binary relations. The most common is the IS-A relation. It is used to construct taxonomies, which are a common way to organize knowledge. This relation is the base for the use of inheritance [10], an efficient inference mechanism. This allows for the common elements of subtypes to be stored only in the super-type, therefore saving effort and space during knowledge encoding.

Another common relation is the Part-Of relation, used to describe structural decomposition [7, 11]. There are several variations of this relation, differing in properties like, for example, transitivity.

With axioms, restrictions can be applied to both relations and concepts. Axioms are written in some formal logic, typically FOL, allowing the use of the full expressive power of that logic. This increases the expressiveness of the ontology and its semantic value, and also what can be inferred from it.

Ontologies serve different purposes. One such purpose, that does not require an inference engine, a knowledge base or even a formal encoding, is to communicate a vision of the world (or of some part of it) between people. For example, they can be used to establish a common vocabulary between working groups from different backgrounds.

In computer systems, ontologies serve several purposes:

- They provide a base structure for knowledge bases and, as mentioned, a reduction in their construction costs by the reuse of already built ontologies and knowledge bases that adhere to them;

- They provide a common vocabulary for communication, either for cooperation between several systems to accomplish some complex task, or to facilitate the interaction between systems of different origins;

- They provide for a means to isolate data from the particular system used to manipulate it, as long as one can build a translator from the system's native storage format to the ontology encoding, enabling interoperability among different systems;

- They can be seen as a way to model the information structure to be used by some software system, during its development.

Ontologies also serve a purpose in the Semantic Web vision [1]. They provide well defined and machine processable meaning to information available online. To this end, online resources must be associated with ontologies by means of resource annotation. Annotation involves both the description of the information according to some ontology and the linking between the description and the original resource.

Having this machine processable information available can benefit both search engines and other automated applications. Search engines can use it to improve information retrieval by distinguishing the meaning and context of the search query. It might also be possible, based on the semantic information, to provide a way in which to combine the search results in a meaningful fashion. Automated agents could use ontologies to communicate and the annotated resources to offer online services that add value by combining information and services already available. For example, to aid in the planning of a vacation trip, combining information from different transport companies, hotels and tourist guides.

2.1.1 Ontology Building

Concerning computer systems, ontology development is a recent field and, as such, methodologies to guide their construction are still a research topic. Notwithstanding there are already some methodologies that share common steps [12–16].

To build an ontology we need, at least, a source of knowledge and some means to extract the knowledge and encode it in the desired formal language. Therefore, the typical "core" steps in ontology building are: knowledge acquisition, conceptualization and formalization/implementation.

Typically, during knowledge acquisition, a set of concepts is produced, along with their natural language definitions. These natural language definitions will serve as a basis to elicit the formal definitions that will form the ontology. Additionally, they will serve to document the intended meaning of the concept names. As the ontology builders are typically not experts in the target domain, this and the next steps should be followed by domain experts. If this is not possible, at least some discussion should take place with them to elicit knowledge and verify that the interpretation is correct.

After completing part of the knowledge acquisition phase[1], it is time to start building the conceptual model – conceptualization. This model defines the meaning of concepts through the relations in which they participate and, maybe, some formal axioms or axiom sketches. The effort here is to encode in relations what is described in the natural language definitions, at least as much as is appropriate to the intended use of the ontology.

Finally, formalization/implementation consists in expressing the conceptual model in a formal language with a particular expressive power. This may require a revision of the conceptual model to conform to the expression constraints imposed by the formal language if these were neglected during the creation of the conceptual model. This step is usually supported by ontology editing tools that provide a way to enter the conceptual model and then write it to a number of formal languages in a user friendly way.

For ontologies that are to be shared and used by groups other than their creators, there must be some assurances about their development and quality. Ontology building is usually done for a specific use and that influences the design decisions made during its development. It follows that to evaluate an ontology for reuse it is helpful to have records of the purpose of the ontology, the choices made during its development and at least the natural language definition of the intended meaning of the captured concepts [17]. This would allow the evaluation of the adequacy of the ontology to the new user's intent. Besides proper documentation, some form of evaluation of the quality of the ontology is also required. These requirements do not correspond to steps, per se, but to activities that must occur throughout the development process. Ontology evaluation in

[1]The knowledge acquisition phase usually overlaps with the creation of the conceptual model.

particular is still a very difficult problem and one without full fledged methodologies [18]. Typically, evaluation is either done by an expert in the target domain or by comparison with another ontology with the same objectives [12]. For the evaluation of the taxonomy, the OntoClean [19] methodology is available. This methodology is based on notions used for ontological analysis in philosophy.

Most, if not all, ontology building methodologies incorporate the steps previously described, in some order. Furthermore, inspiration from software development methodologies can be used to enhance and create ontology building methodologies by replacing the traditional steps of design, implementation, testing and documentation, with there counterparts in ontology building.

2.2 Ontology Learning Methods

The construction of software agent programs that communicate using ontologies as a basic vocabulary and the increasing effort to make the Semantic Web vision a reality put pressure on ontology development methods. Automatic and semi-automatic ontology learning methods offer hope of reducing the effort and cost of ontology development.

The area of semi-automatic ontology learning is very recent, with the first two workshops occurring at the 14th European Conference on Artificial Intelligence in 2000 and 17th International Joint Conferences on Artificial Intelligence in 2001. It combines the efforts of machine learning methods and of natural language to automate knowledge acquisition and conceptual model creation.

The steps involved in learning the conceptual model are, in an very coarse view, similar to that of manual learning. First some knowledge sources must be chosen, from which a set of concept candidates, or instances of potential concepts, are retrieved. From these candidates, a subset is selected for use in the conceptual model. These concepts are then linked by relations inferred using information extracted from the knowledge sources. It is typical to distinguish between the inference of taxonomic relations and other relations.

As to formalization, if the learning method already works in a formal language then there is no need for a separate formalization step. If this is not the case, then the formalization step is reduced to the automatic transformation of the internal model used by the learning method to one or more formal languages, in a similar way to that of manual ontology editors.

These methods, typically, are not able to completely automate the ontology development process. Usually axiom inference is not done and some, if not all, concepts and relations may have to be named by hand. Ontology evaluation is still a difficult problem, even for manual construction, which means that the results are difficult to assess.

As a positive side effect of semi-automatic ontology learning, the effort to fill a knowledge base (instance population) that follows the learned ontology could be significantly reduced. This results from using the models obtained while learning the ontology to extract information from sources compatible with those used during the learning process.

As in the case of manual construction of ontologies, one of the very first steps is the selection of the knowledge sources. The choice of the source, such as text, dictionaries, relational schemas or other semi-structured data, will have an impact in the learning methods available.

The next few sections briefly explore the available methods for learning from text (the most common source) and from semi-structured data (the focus of this thesis).

2.2.1 Learning From Text

Manual ontology construction is mostly done from texts, which are the most common source for information. As such, it is no surprise that ontology learning has mostly followed the same path and that learning from text is the most explored area in ontology learning.

Several techniques exist for the extraction or inference of each component of an ontology from relevant texts, as well as several approaches to the overall problem. A description of the main techniques for each ontology component is now given, and a summary of the common approaches to the problem will follow.

Common Techniques

Concept Candidate Extraction The first step to learn an ontology is to identify a set of concept candidates from the available text corpus. The word "term" will be used to refer to a single word or short sequences of words, such as noun-phrases, that correspond to a concept.

A simple solution, as adopted in [20], to the identification of possible terms is to collect all words except those present in a stop-list or, word sequences between stop-list words. The stop-list would contain a set of verbs, conjunctions, personal pronouns and prepositions that are to be disregarded when searching for terms. The main difficulty is, of course, to build such a list.

A more common approach consists in the use of a Part-of-Speech (POS) tagger, that attempts to identify and label the different POS in the text. From this, the set of nouns and noun-phrases can be used as terms. Although some errors can be introduced due to language ambiguity, the results should be better than those obtained using only a stop-list. However this is not always available for all languages and may require training the

POS tagger to the specific language.

As in most cases morphological variants of words have similar semantic interpretations, it could be desirable to apply a stemming algorithm to the terms. Also used are lemmatization algorithms, where inflectional and variant forms of a word are reduced to their lemma: their base form, or dictionary look-up form. Besides reducing the number of total concept candidates (and providing for larger frequency counts for further processing), this type of processing produces more adequate names for concepts.

Technical documents make abundant use of acronyms and abbreviations, making their identification important. In [21], an approach involving the use of manually crafted extraction patterns was used to extract acronyms and their expansions from free text. This processing brings similar advantages as the application of stemming and lemmatization.

Sometimes there is interest not only in obtaining concept candidates, but also instances and their relation to concepts, as shown in [22], for example to detect dates, monetary amounts, emails, etc. That can be done using specialized data patterns, but may also require some disambiguation. For example the string "17.7 million" is clearly a number, but it is not clear what it quantifies. Numeric values, after adequate disambiguation can be useful in frequency counts for relation extraction and the inference of value range restrictions.

Finally, sometimes the set of concepts does not correspond to terms in the text but to sets of documents in the corpus. In [23], documents are clustered on the basis of the similarity of their content (word frequency) and a topic label is assigned from within a predefined set of topics. For each topic a set of documents must be supplied to train the algorithms used. A similar approach makes use of latent semantic indexing, where documents are clustered together, given the relative frequency of the words they share.

Concept Selection The resulting list of terms obtained during the extraction step is usually too large, containing terms that are not very interesting to the problem at hand. So there is a need for techniques to reduce such a list.

One straightforward way of making term selection is to use frequency counts in the corpus and choosing the most frequent terms, given a predefined threshold. However, some care must be taken not to select terms that are too general (high frequency count) and too specific (very low frequency count).

One way to do that is to use the Term Frequency Inverted Document Frequency (TFIDF) measure, defined with the help of: *term frequency* – $tf_{t,d}$ – that counts the number of times the term t occurs in the document $d \in \mathcal{C}$, where $\mathcal{C}$ is the corpus; *document*

frequency – df_t – that counts the number of documents in which the term t occurs.

$$tfidf_t = \sum_{d \in \mathcal{C}} tf_{t,d} * \log \frac{|\mathcal{C}|}{df_t}$$

TFIDF weights the frequency of a term in a document with a factor that reduces its importance if it appears in the majority of the documents.

Another approach, described in [20], is to use a distance measure to select the terms to include as concepts. Selected terms are those that are closer than some predefined threshold to a concept already in the ontology. This approach could be used for either ontology enrichment or incremental building of the ontology.

The distance measure is based on two sources:

- Co-occurrence of words, that is, whether two words appear together in the text, according to some rule. Typical rules are 'within a maximum distance of 5 words' or 'within the same sentence'. To make use of this information, a collocator vector is built. This vector counts the frequency of co-occurrence of a particular term with an ordered set of words. The set of words used in the vector are the n most frequent words that co-occur with at least two concepts in the ontology.

- Base distance measure between concepts in the ontology. This base distance can be computed in many ways. One possible approach is to use the minimum number of steps S, counted as transposed relations, to get from one concept to the other. This could be formulated as $d(x, y) = e^S$, where x and y are concepts and e is Euler's constant.

Using this information, a set of weights is derived for the collocation vector of each concept such that the distance between concepts (as given by the collocation vectors) is as close as possible to the actual ontology structure. These weights are then used to measure the distance between the new term and the already existing concepts in the ontology.

Another possible case for concept selection is the one that occurs when re-using a more general ontology. In this situation, instead of a frequency measure, one simply selects the terms for which there is a match in the general ontology, obtaining that way a subset specific to the target domain. However, it is likely that the general ontology did not contain enough detail and so concepts specific to the target application may be missing. To correct this, either of the previously discussed approaches could be used to select additional concepts and enrich the resulting ontology.

A different path to the selection of which concepts to add to an ontology is presented in [24]. There, the terms selected to become concepts are the direct result of restricting the extraction procedure to the phrases that contain a set of seed words. These seed

words are either given by the user, at the beginning of the process, or result from the terms found in the phrases of a previous iteration of the process.

In some cases a term can have different semantic meanings in different parts of the corpus. If more than one of the meanings is relevant to the ontology domain, then care must be taken to disambiguate between the different senses. Taking advantage of WordNet, [25] presents a sense disambiguation approach using topic signatures.

A topic signature is a list of words that co-occur with a concept (or word sense), in some context, and their frequencies. The basic idea is that when a given concept (or word sense) appears in the text, most of the same words appear with it. In [25], these topic signatures are built using documents retrieved from the WWW. The documents are retrieved using queries formed with the information in WordNet for each sense of the term. The context used for each word sense is the collection of documents retrieved for it.

To select the correct word sense in some part of the corpus, a window is selected around the term occurrence in the corpus. From the words in this window, and a similarity function, value is obtained for each topic signature. The sense chosen is the one that has the highest similarity value.

Hierarchy Inference The most commonly inferred relation is the IS-A relation, that serves as a backbone hierarchy to most ontologies.

Using a concept similarity measure, based on the corpus, it is possible to apply hierarchical clustering methods to obtain the IS-A relations. These can be either top-down or bottom-up. A top-down algorithm starts with a single cluster containing all the concepts and then in each iteration splits the less coherent cluster. Cluster coherence is a measure of how similar its elements are. A bottom-up algorithm will do the reverse, starting with a set of single element clusters, one for each concept, and, at each iteration, merging the more similar clusters.

The most important similarity measures between concepts [12] are:

- the *cosine difference* between two concepts x and y is given by

$$cos(\vec{x}, \vec{y}) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2 \sum_i y_i^2}},$$

  where $\vec{x}$ and $\vec{y}$ are the co-occurrence frequency vectors for concepts in the corpus, and x_i and y_i are the frequency with which x and y co-occur, respectively, with the concept ith concept in the vector;

- the *Kullback-Leibler divergence*, between two concepts y and z is given by

$$D(p_y||p_z) = \sum_{x \in X} p_y(x) \log \frac{p_y(x)}{p_z(x)},$$

where X is the set of all concepts, $p_y(x)$ and $p_z(x)$ are the probability estimate that x occurs, given that y or z occur, respectively. The probability estimates are obtained from the co-occurrence frequencies.

When concepts correspond to document sets, a more sophisticated approach using a variation of the Self Organizing Tree Algorithm (SOTA)[26] can be used, as shown in [23]. This algorithm distinguishes between leaf nodes (named cells) and interior nodes, and works as follows. It starts with one interior node with two leafs and iterates between two phases: distribution and expansion. In the distribution phase, a subtree is selected, rooted at a K level ancestor of the parent of the new cells. The input data (documents in this case) is distributed among the cells in this subtree. In the expansion phase, a value is assigned to each cell and if the value exceeds a certain threshold the cell is selected for expansion. The selected cell is converted into an interior node with two newly created cells as children. When all cells have values bellow the threshold the iterations stop and the tree is pruned to remove all empty cells.

The hierarchy inference methods just described result in binary hierarchies with unnamed interior nodes. Two possible approaches to name these interior concept nodes are to use an existing ontology or to have a person name them by hand.

To use an existing ontology, as shown in [12, 23], the first step is to map the concepts of our new ontology, of which we know the labels (leaf concepts of the hierarchy), to concepts in the existing ontology. Here a domain ontology or a general ontology, such as WordNet, could be used. Afterwards, the interior nodes are labeled in a bottom-up fashion by following the IS-A relations in the existing ontology. To name an interior node in the new ontology, whose children are already labeled, we use the label of the nearest common ancestor in the existing ontology.

When the work of naming the interior concept nodes is left up to a person, then it could be helpful to show some information to guide this work. Besides the hierarchy itself, it could be helpful to present excerpts of occurrences of the child concepts in the original corpus.

A more incremental approach to building the hierarchy is to add one concept at a time to the already existing hierarchy. In [27] a simple algorithm, similar to a greedy search is used. Starting with the most general concept, the concept to be added is compared with the current concept and with its direct descendants by means of a similarity measure based on topic signatures. If the concept is closer to the current, then it is added as one of its children. Otherwise the process is repeated with the closest child, in terms of the similarity measure. Later, in [28], the similarity measure is improved by combining different types of semantic signatures besides the usual topic signature. Furthermore, in [29] the similarity measure is further improved by combining it with information given by

lexico-syntactic patterns. In a similar approach, in [20], each new concept is placed as a child of the concept with which it has the smallest distance. This is the same distance measure, obtained from word collocation and a base distance derived from the ontology, that was described regarding concept selection.

Instead of using frequency based approaches to infer IS-A relations, these can be lifted directly from the corpus by the use of lexico-syntactic patterns, as shown in [30]. Lexico-syntactic patterns are pattern rules that use both words and syntax elements, that are used to recognize relations in texts. For example, the pattern "NP {,NP}* {,} *or other* NP", where NP is a noun-phrase, can be used to recognize the relations *bruise IS-A injury*, *wound IS-A injury* and *broken bone IS-A injury* from the text excerpt "*Bruises, wounds, broken bones or other injuries ...*". Lexico-syntactic patterns are sometimes called Hearst patterns, as Hearst introduced a set of patterns for the acquisition of hyponym (IS-A) relations [30]. Note that, when using only the IS-A relations extracted in this manner, there is no guaranty that the result will be a single tree. Another type of pattern that can be used, as shown in [31], is the text title hierarchy. Here, one takes advantage of the hierarchy between title, section title, subsection title, etc. to define relations between the terms that appear in those titles.

These methods can produce non-binary hierarchies, as opposed to hierarchy clustering methods or SOTA, giving them a more expressive solution space, closer to what would result from manual ontology building.

A system capable of deeper linguistic analysis enables an approach closer to natural language understanding. In [32], when a new concept is to be added to the ontology, an hypothesis space is built from the information present in the text. The hypotheses are eliminated comparing the information from the text with the information about the concepts already in the ontology. To that effect, a quality measure is associated with each hypothesis based on two components. The first component is the linguistic quality. Not all linguistic constructions provide the same level of certainty about their interpretation. Some provide precise information, such as an apposition "the printer Xpto...", while others provide less information. This variation allows some hypotheses to be considered as more likely than others, contributing that way for the quality measure. The other component of the quality measure uses the concept description of concepts already in the ontology. It measures how closely the hypothesis matches the existing ontology structure and favors hypothesis that better resemble the existing concepts.

If a general ontology, such as WordNet, is available, a different approach to hierarchy inference is possible as long as a mapping is available between the concepts to be used in the ontology and those present in the general ontology. In such a case, the IS-A relations are given by taking the shortest path along IS-A type links on the general ontology between

each pair of concepts that are to be included in the new ontology.

A more manual approach comes from building a differential description of the ontology, presented in [21]. The differential approach is based on the handmade description of concepts by enumerating what distinguishes them from their parents and siblings in the hierarchy. This guides the manual construction of the base hierarchy. Afterwards, individual instances are manually assigned to each concept providing their extensional description[2]. Using this information, the hierarchy can be extended through set operations, where a subsumption relation corresponds to set inclusion.

Non-Taxonomic Relation Inference Non taxonomic relations seem to have been less extensively studied, perhaps for being a more difficult task without the knowledge of what type of relation is being extracted.

Inspired in *database mining*, one approach to non-taxonomic relation inference is the use of association rules. Association rules are simple if-then rules inferred from a set of transactions, that is, a set of lists of items that occur together. In our case, learning non-taxonomic relations, we are interested in rules where both the antecedent (the "if" part) and the consequent (the "then" part) are a single item. The items are the concepts of the ontology and the transaction list is their co-occurrence, for example at the sentence level, in the corpus. An inferred rule $X \rightarrow Y$ will be taken as a binary relation from concept X to concept Y.

Selection of candidate rules $X \rightarrow Y$ is based on the notion of support (percentage of transactions that contain $X \cup Y$) and confidence (percentage of transactions that contain Y from the set of transactions that contain X). From the possible rules, given the set of concepts, those that have a greater support than a predefined minimum are selected. This set of rules is then further refined by selecting those that have a confidence greater than a predefined minimum and these are inserted as binary relations in the ontology.

Note that relations where the source and target domain are the same cannot be inferred from the direct application of association rule inference methods, as the co-occurrence of the concept with itself is not considered. To enable the inference of this type of rules, one possibility is to create an new artificial concept that takes the place of the second occurrence of the target concept in transaction. This artificial concept is then replaced by the original concept when adding the rules to the ontology.

Concepts in high levels of the ontology's hierarchy may have very few direct instances, making it difficult to apply association rule learning. To solve this issue, occurrences of child concepts are counted as occurrences of their parents. This then enables the inference of rules at higher, more abstract, levels of the hierarchy. Furthermore, when an

[2]An extensional description of a concept is the set of all its instances.

inferred rule is present both between concepts at a high level in the hierarchy and in their descendents it is then possible to simplify the ontology by pruning the ontology of the subsumed lower level relations.

When re-using an existing domain ontology or a more general ontology such as WordNet, a possible approach [33] (as with the case of IS-A relations) is to add a relation between two concepts, if there is a relation between them, in the existing source ontology. Nonetheless it is not always the case that two concepts would be directly connected in the source ontology, in which case the path between them in, following non IS-A relations, is taken. Note that this does not always produce good results. To improve results not all paths are accepted. Those that have a number of steps larger than a predefined threshold are excluded.

As with hierarchical relations, lexico-syntactic patterns can also be of assistance in retrieving relations between concepts [24].

These methods cannot, for the most part, supply a name for the relation, leading up to the use of manual labeling or an attempt to rely on a general ontology such as WordNet to provide for a label.

Learning Approaches

There are several ways to approach the problem of ontology learning from text:

- Enrichment – Take an already existing domain ontology and extend it with new concepts and relations extracted from a corpus.

- Build from scratch – take a corpus about the target domain and build the ontology using the steps and techniques described before. The construction may be guided by a changing set of seed words [24] or even be a part of a larger cycle where the ontology is evaluated and improved at each execution cycle [34].

- Specialization – Take a general ontology such as WordNet or a domain ontology with a broader scope than the target domain and, with the help of a focused corpus, reduce the ontology to the concepts present in the corpus [33]. This may be followed by an enrichment step if the source ontology doesn't cover the corpus completely.

- Merge – Although not within the scope of this overview, an ontology could be built by merging existing domain ontologies. As an example of the methods available to perform ontology merge, there is the FCA-Merge [35] procedure (concept mapping is done using sets of documents to distinguish the concepts and applying Formal Concept Analysis), PROMPT [36] (concept mapping is done in a semi-automatic manner, where the system proposes concepts to be merged and partially automates

some steps) and Chimaera [37] (edit and merge tool that aids users through heuristic selection of merge candidates) and many others.

2.2.2 Learning From Structured and Semi-Structured Data

Compared to learning from text, learning from structured and semi-structured data is a field with less work done. The idea is to learn, not from texts about the domain, but from actual samples from the domain.

The major approaches here are:

- Inference of new ontology information from existing resources – Two examples are the work on extraction of new concepts from Resource Description Framework (RDF) annotated resources [38] and the inference of a taxonomy of resources from their metadata [39]. The method reported in [38] enables the extraction of new concepts from RDF graphs. The concepts are defined both by their extension, as given by the set of RDF resources that are examples of the concept, and by their intention, as given by the generalized graph that matches each of the graphs that correspond to the resources in the concepts extension.

  The process works by collecting RDF triples (resource, property, value) and recursively generalizing pairs of triples. Triple generalization is done by replacing the value part. If the values are RDF classes, then the existing hierarchy relations are used to find a common ancestor to the values and that is used as a generalization. However, when no such class is available, a new anonymous one is used. Generalized triples are joined together if they share the same extension, removing triples that subsume others, to keep the new concepts the most specific they can be, avoiding over generalization. Taxonomy relations between the new concepts are simply the result of a subsumption between their intention graphs. Note that RDF allows for multiple inheritance.

  To avoid problems with exponential complexity, each resource graph is defined at first as the largest connected subgraph of length one. The method proceeds to iterate over the length of the description, incrementing it at each step. The graph of each incremental step is built reusing the result of the previous step.

  In [39], a method is reported for the inference of a taxonomy of Learning Resources. Each resource is described by Extensible Markup Language (XML) metadata and a set of keywords is extracted from the metadata. Keyword extraction is done in a way similar to the one used in methods for ontology learning from text, by using techniques such as lemmatization, Part-Of-Speech tagging, etc. To obtain the taxonomy, a clustering algorithm is employed [40]. This algorithm has the

particularity that clusters may overlap. Finally, to describe each cluster that will correspond to a new concept in the taxonomy, a subset of the keywords is chosen. Those keywords that have a larger squared frequency in the cluster than in the entire dataset, by a predefined amount, are selected.

- Mapping existing schemas to existing or new ontologies – Two examples are [41], where a method is reported to automatically map diverse data sources, each with a fixed schema, to a mediator ontology, and [42] that reports methods to transform existing schemas, such as relational database schema, XML Schema, XML Document Type Definition (DTD) and Unified Modeling Language (UML) class definitions to ontologies based on a transformation to regular tree languages [43].

  In [41], the problem of matching data in different, but fixed, schemas to a mediator ontology is approached. To accomplish that in an automatic manner, a group of four learners is combined: a nearest neighbor classifier based on TFIDF called Whirl, a Naive Bayes classifier, a name matcher (e.g. price, house_location) and a county-name recognizer. The information from the learners is combined using a meta-learner. For each of the learners, the meta-learner learns how much weight to associate with each of the label values returned by that learner. For each element in the source schema the system then chooses the label that is given to the highest number of instances as long as the percentage is above a predefined threshold and the difference in the number of instances between the best label and the second best choice is larger than another predefined threshold. Otherwise it reports a failure.

2.3 Wrapper Induction

One area that has close resemblance to this work is web page wrapper extraction. Web page wrappers are programs written to extract information from web pages, enabling further manipulation, such as data integration from various sources.

Manual creation of wrappers is known to be problematic, mainly because wrappers are difficult to write and to maintain due to frequent changes in web page format. As a result, several methods have been developed to automate this process, surveyed in [44].

Automatic (or semi-automatic) wrapper generation relies on web pages being generated from a structured source (usually a database) in a consistent and automatic manner. The problem is then to identify what the original structure was, or at least the main fields of interest, and to do so in a manner as automatic as possible to be able to cope with changes in web pages by quickly regenerating the wrapper.

There are several approaches to wrapper induction, and a possible classification [44]

is:

- Declarative languages – languages developed for the sole purpose of creating wrappers. These languages facilitate web wrapper development by supplying language primitives and function libraries built specifically for the task. The wrapper itself must, however, be built by hand.

- HTML aware tools – these tools base identification of the source structure or of the interesting fields on the presence of HTML tags.

- Natural language aware tools – these tools use the available methods for natural language processing (NLP) to extract information from web pages. After some processing, such as part-of-speech tagging and lexical semantic tagging, etc., relationships are built between phrases and sentence elements, allowing for the construction of rules based on that information. These are of particular use in web pages where the information to be extracted is shown as blocks of text, as in, job listings, apartment advertisements, etc.

- Machine learning approaches – these tools take as input pages where the fields of interest have been labeled. From the input, a wrapper is inferred that conforms with a pre-defined model and incorporates the labeled fields. The model can range from a flat tuple list to more complex structures such as finite state automata.

- Modeling based tools – these tools take as input the structure of the target objects, as defined by a set of modeling primitives (e.g. tuples, lists, etc.), and then search web pages for instances of those structures (portions that implicitly comply to those structures).

- Ontology based tools – these tools use a hand-crafted ontology to locate and retrieve data from web pages. If sufficient information is coded in the ontology it is possible to map web page content to ontology concepts. This leads to a more robust way to accommodate changes in web page structure. This happens because it is no longer required that the web pages follow a strict structure, as long as the mapping to the ontology remains possible.

As the source documents for this work are HTML pages, the most relevant group of wrapper generation approaches is the group of HTML-aware tools.

All these tools have at least one common step, that is cleaning the HTML of the source documents. Web pages are known for commonly exhibiting syntax errors due to the tolerance that web browsers have for such mistakes. So, as a first step, heuristics are employed to fix as much as possible the errors present in these web pages. This also has

a secondary effect, as it allows the use of Document Structure Model (DOM) [45] tools, facilitating their manipulation.

One approach to the problem, used by HTML-aware tools, is to use HTML tags, along with plain text, in the identification of relevant fields. For example, in [46], a method is proposed to build regular expressions to extract field values using occurring tokens on either side of the field. These tokens are either HTML tags or specific words.

A more global approach is also possible, using the entire structure of the web page to infer the presence of relevant fields. RoadRunner [47], generates wrappers by looking at several examples from the same web site. The structure of these examples is compared in order to find a common pattern. It can infer the location of fields, optional patterns and iterators.

A more user-guided approach is exemplified by XWRAP [48]. Here, the first step is performed by the user that graphically selects the regions of interest. Then, with the aid of the user, significant semantic tokens (the field names of interest) are selected and from those the value locations are identified. Finally, using heuristics based on font size changes and on the structure of the page (nesting of tables, paragraphs, lists, etc.) the hierarchical structure is constructed. These steps result in a set of declarative rules that are then transformed into an executable program re-using the component library supplied. The information extracted by the wrapper is stored in XML files.

A tool that could be seen as a declarative language is the W4F (World Wide Web Wrapper Framework) [49]. It proposes a language, and supporting framework, to accomplish four tasks in wrapper creation: web page retrieval, cleaning, information extraction and mapping into user-defined data-structures. The extraction rules are based on the HTML tags along with the use of regular expressions and other capabilities.

2.4 Regular Language Learning

Given a finite set of symbols Σ, the alphabet, a formal language is a finite or infinite set of strings built with symbols from Σ. This set may or may not contain the empty string, usually referred by the symbol ϵ.

Chomsky characterized the languages recognized by grammars into four classes [50], each a proper superset of the next. These are often called the Chomsky hierarchy and can be seen in Table 2.1, where each type is shown followed by the grammar and automaton required to recognize it. A definition for each language, grammar and automaton can be found in [51].

A description of regular languages and a brief overview of methods for regular language learning can be found in the following.

Table 2.1: Automata theory: formal languages and formal grammars

Chomsky hierarchy	Grammars	Languages	Minimal automaton
Type-0	(unrestricted)	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Context-sensitive	Linear-bounded
Type-2	Context-free	Context-free	Pushdown
Type-3	Regular	Regular	Finite

2.4.1 Regular Languages

There are several classes of formal languages. Important among them is the class of regular languages. A regular language can be defined by induction as follows:

- The empty set, $\emptyset$ is a regular language;
- Let ϵ be the empty string. Then $\{\epsilon\}$ is a regular language;
- If $s \in \Sigma$ then $L = \{s\}$ is a regular language;
- If L_1 and L_2 are regular languages then $L = L_1 \cup L_2$ is a regular language;
- If L_1 and L_2 are regular languages then $L = L_1 \cdot L_2 = \{ab | a \in L_1, b \in L_2\}$ is a regular language;
- Let $L^0 = \{\epsilon\}$, $L^i = L \cdot L^{i-1}$ for $i \geq 1$, then the Kleene closure of L, denoted L^*, is the set $L^* = \cup_{i=0}^{\infty} L^i$. The positive closure of L, denoted L^+, is the set $L^+ = \cup_{i=1}^{\infty} L^i$. Both are regular languages.

One way to express regular languages is by the use of regular expressions, that are defined by induction in a similar manner to regular languages:

- $\emptyset$ is a regular expression;
- ϵ is a regular expression and stands for the set $\{\epsilon\}$;
- For $a \in \Sigma$, a is a regular expression and stands for the set $\{a\}$;
- If r and s are regular expressions, then $r + s$, rs, r^* and r^+ are regular expressions standing for union, concatenation, Kleene closure and positive closure, respectively.

There are other ways to represent regular languages. Another way resorts to the use of Deterministic Finite Automata (DFA). A DFA is defined as a tuple $D = (Q, \Sigma, q_0, F, \delta)$, where:

- Q is a finite set of states;
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is the set of final or accepting states;
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function.

The function δ is extended to $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$, as follows:

- $\hat{\delta}(q, \epsilon) = q$;
- $\hat{\delta}(q, wa) = \hat{\delta}(\delta(q, w), a)$, where $a \in \Sigma$ and w is a string defined over Σ;

A string w is accepted by the DFA D iff $\hat{\delta}(q_0, w) \in F$. The language corresponding to a DFA is the set of strings accepted by it.

Given a finite set of strings, one way to build a DFA that accepts it is to build a Prefix Tree Acceptor (PTA). A PTA is built starting with an initial state with no transitions, no accepting states and no defined transitions. For each string, start with the initial state and proceed for each symbol of the string by either following the transition by that symbol to the next state, or, if that transition does not exist, adding the transition and a new state. The states where each string ends at, are added to the set of accepting states.

DFA are a type of Finite State Automata, where for each state and each symbol, there is only one possible transition. Another, that can accept the same class of languages, is the Non-Deterministic Finite State Automaton (NFA), where for each state and each symbol there can be more than one possible transition to another state. There can even be transitions on the empty string symbol.

Yet another equivalent formalism is a regular grammar. Grammars are defined by a tuple $G = (N, \Sigma, P, S)$, where N is a non-empty set of non-terminal symbols, Σ is the alphabet, P is the set of production rules and $S \in N$ is the start symbol. A production rule is a rule that specifies a way to replace a non-terminal symbol in a string. A string is accepted by a grammar if there is a sequence of applications of production rules that, starting with the start symbol, result in the given string.

There are several ways to specify grammars that recognize regular languages. One way uses right-regular languages, where the allowed production rules are of the form:

- $A \rightarrow a$, where $A \in N$ and $a \in \Sigma$;
- $A \rightarrow aB$, where $A, B \in N$ and $a \in \Sigma$;
- $A \rightarrow \epsilon$, where $A \in N$ and ϵ is the empty string.

.

2.4.2 Regular Language Learning Approaches

For the method presented in this work, it is necessary to infer a regular language only from positive examples, so a brief review of other methods for learning regular languages is presented here.

Gold [52] shows, under the model of language identification in the limit, that regular languages cannot be learned using only positive examples. In this model, the learner is presented a unit of information about the language (for example, a string belonging to the language) at each time step and makes a guess as to what the language is. A language is learned, if after some finite amount of time, all of the guesses after that time are identical and correct.

Despite this result, if some restrictions are applied, it is possible to infer some language classes. In [53], Angluin introduces the class of k-reversible languages, a subclass of regular languages that is learnable using only positive examples. In broad terms, the algorithm starts with a prefix-tree acceptor and merges states so that the resulting Deterministic Finite Automata (DFA) is a k-reversible language. That is, a merge is done if the current DFA breaks some property of k-reversible languages.

Another way to approach the problem is to use additional information along with the positive examples. Sakakibara in [54, 55], proposes an approach to infer context-free languages, a proper superset of regular languages, using structured strings. Structured string are strings with some parenthesis added to encode the shape of the parse tree, providing a restriction to the form of the grammar that could have generated the examples.

Query learning, sometimes called active learning, provides an alternative to using only positive examples to learn a regular language. With query learning, the learner algorithm is allowed to make queries or to experiment with the unknown target language. Based on the approach described by Gold [56], Angluin proposed the L* algorithm [57], which can learn a Deterministic Finite Automaton (DFA) in polynomial time. This algorithm is described in more detail in Section 3.2.1.

Schapire, in [58], presents an alternative approach that does not require the availability of a reset signal to take the automaton to a known state. In the previously described approaches, the information provided to the algorithm is a set of strings, where, for each string, the DFA is to be traversed starting from the initial state. The set of strings can be seen as a single sequence resulting from the concatenation of all the strings, using some symbol as a separator. This symbol would be the reset signal, taking the DFA from whatever state it is in when the symbol is read to the known initial state. Schapire's algorithm does not require this and takes as input a single sequence of symbols.

In query learning, one type of query is the equivalence query. In an equivalence query, a solution is presented to the teacher, or oracle, that either accepts it as being equivalent

to the target goal or returns a counter-example that distinguishes both. Parekth [59] shows that equivalence queries can be avoided if a structurally complete set is available. A structurally complete set is a set that covers every state transition at least once and uses every accepting state at least once. The method uses the structurally complete set to build a lattice of Finite State Automata (FSA), and then uses membership queries to prune the lattice until only one FSA is available.

If, besides positive examples, negative examples are also available, then there is a wealth of algorithms that can learn regular languages efficiently. Some reviews of these works can be found in [60, 61].

2.5 Relation to this Thesis

As already stated in the introduction, this thesis concerns the development of a method to learn ontologies about document structure, that is, ontologies that capture the implicit internal structure of a particular document type.

As such, it falls into the category of ontology learning methods, and most likely in the group of ontology learning methods from semi-structured data. It can be viewed as learning from semi-structured data because it takes its information not from the actual text of the documents, although some clues come from there, but mainly from the overall structure of the document and in particular from the graphical presentation properties encoded in its HTML code.

The extraction of information from web pages is the subject matter of the Wrapper Induction work, although their purpose is not the inference of an ontology, but to enable a database like use of that information. In particular, HTML aware wrapper induction approaches bear similarities to the first step of the method developed in this thesis, described in Section 3.1. In particular, they share the preprocessing of HTML pages with the objective of cleaning frequent syntax errors and transformation into valid XML syntax (using the XHTML format) to ease further processing.

A particular document type is more than just a sequence of concept instances, requiring some effort to infer its structure. As such, this work applies a regular language learning approach to infer such structure, described in Section 3.2, that is later to be encoded into an ontology, as by the work shown in Section 3.3. In particular, it builds on the existing active learning approaches by using the L* algorithm and expands the inference of Deterministic Finite Automata from positive examples, by the application of the Minimum Description Length principle.

Chapter 3

Learning Document Structure Ontologies

Automatic or semi-automatic ontology learning can be seen as important means with which the ontologies necessary for the Semantic Web vision could be created. Such methods would reduce the burden of creating an ontology, a costly and time consuming task, and enable the extraction and use of the information already present in the World Wide Web, bringing the Semantic Web vision closer to reality. Moreover, the conversion, or annotation, of the vast number of existing resources is only practical if the process can be somehow automated.

The bulk of ontology learning work has been done in the extraction of information from natural language text, as this is the richest source for information and also the main source for manual ontology construction. The use of semi-structured data as a source for ontology learning has focused on the identification and abstraction of local patterns in the sources or in a mapping from a well defined structure input, such as database schemas.

Besides the work focused on ontology learning, there are works on so called wrapper induction, that intend on the extraction of information from web pages that share a common schema. However, most of this work relies on the fact that these pages are automatically generated from databases and therefore present a very regular structure.

The focus of this work is on semi-structured documents, where an implicit common structure exists, but the instances are not computer generated. This makes it more difficult to identify that structure due to it not being expressed exactly in the same manner in all instances.

Documents are present in a variety of file formats, some being easier to manipulate than others. This work has focused on technical papers stored in HTML format. There are many examples of these throughout the Internet and they can be parsed without much trouble using freely available tools, making these a good target.

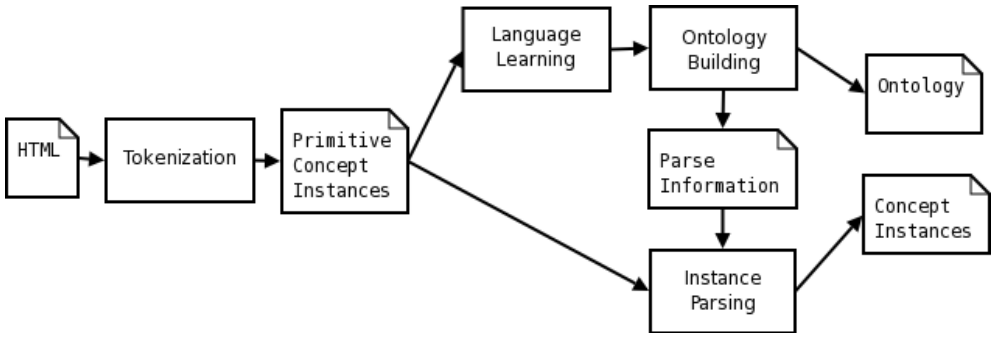

Figure 3.1: Ontology Learning Process

As stated in the previous chapters, the target for the learning process is to infer an ontology describing the structure of a set of documents. The process, shown in Fig. 3.1, was divided into three parts: (1) identification of a set of primitive concepts (tokenization); (2) inference of a language using these concepts as an alphabet that covers the source documents (language learning); (3) extraction of an ontology description from that language (ontology building). Each of the next sections describes a step in this process.

With the information obtained during the process of learning the ontology, it is also possible to parse the documents and create instances of the concepts in the final ontology (or use the information to annotate the source documents with the corresponding concepts from the ontology, which amounts to the same problem). Parsing will be mentioned in the section describing the ontology building step.

3.1 Tokenization

The first step in the ontology learning method is to map the content of the HTML source documents into a set of primitive concepts. These concepts will be used later as the input for the next step of the ontology learning process.

The overall approach to this task is shown in Fig. 3.2, and consists of three steps: (1) pre-processing, (2) segmentation and (3) classification. The results are stored in a XML file for each source document, where along with each instance text, the concept name and attributes about the instance are also stored. These attributes will be detailed in Section 3.1.1.

The pre-processing step, shown in the diagram, covers the transformation from HTML to XHTML and fixing, as much as possible, the errors present in the source document. As is well known, HTML documents are very likely to have errors, failing to conform with W3C standards, mostly due to the high tolerance that current web browsers have for illformed pages. Besides aiming at the correction of these errors, the transformation into

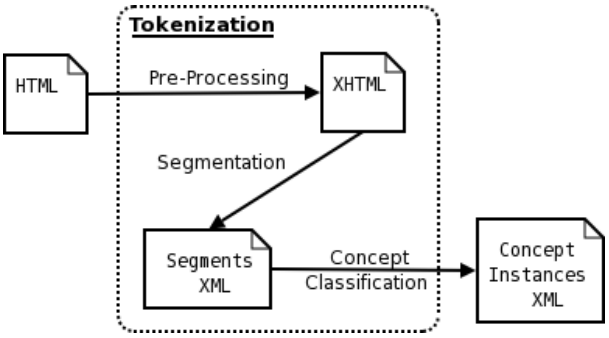

Figure 3.2: Tokenization Overview

XHTML allows us to leverage the existing tools for XML processing. To accomplish the requirements of the pre-processing step, the HTML Tidy[1] tool was used. This program uses a set of heuristics to try to correct the mistakes found on HTML pages and allows the transformation of HTML pages into XHTML. The program is not always able to correct the input documents, but works for the vast majority. Documents for which the HTML Tidy was ineffective were not considered in the experiments as they cannot be used in conjunction with XML processing tools and would require either extensive manual corrections or heuristics to recover their content, which is beyond the scope of this work.

3.1.1 Document Segmentation

The segmentation approach is based on the assumption that when a concept instance appears in a document, that is, when a structural change occurs, it will be visible through formating hints or visual clues. It is further assumed that the concept instances cover, in terms of processable text, the whole document, and that the primitive concept instances do not overlap. In the following descriptions concept instances will be referred to as segments if they are not yet associated with a particular concept name.

The source documents, XHTML pages, are trees of tags, with the text placed at the leaves. Many of these tags will be of interest for the extraction of visual clues to segmentation. To organize and select information from these tags and prepare for the classification stage, a set of attributes were chosen to describe each segment. Segments will therefore be composed of text and a set of attribute value pairs.

The set of attributes is divided into two main groups:

1. XHTML based attributes - computed from the inspection of XHTML tags. These can further be divided into:

   (a) Segment divider - used to drive the segmentation stage.

[1]Available at http://www.w3.org/People/Raggett/tidy/

(b) Non-segment divider.

2. Post-Processing - computed after segmentation, either using their segment content, context or classification.

In general, the attributes give information about three features of segments:

Text Format These are the attributes that drove the segmentation. In our opinion, these attributes are one of the main means by which people identify the different parts of an article.

Font size (segment divider, post-processing) The current font size. The font size attribute is normalized by dividing it's value by the maximum found in the current article and then converting the result into an integer from 1 to 10. This allows for a better comparison between articles that use a different scale of font sizes on section titles, for example.

Font style (segment divider) The current font style, that is, if the text is in bold, italic, etc.

Text Information A set of attributes that reflect information contained in the segments text. All of them are post-processing attributes.

Text size The length of the text in each segment has a big weight on its classification. This information is transformed into the following discrete scale: 0 to 30, 30 to 80, 80 to 160, 160 to 1600, 1600 to 16000, bigger than 16000. This scale was obtained empirically. Initially, the first three intervals were collapsed and where intended to capture the size of a one or two line sentence, for example the title of an article. Later it was observed that it was helpful to further subdivide this interval, so that small segments of text where not mistaken as titles and section titles. The other intervals are there to capture small paragraphs and larger segments of text.

Keyword If the text segment starts with one of a given list of keywords, the attribute value is the keyword index in the list.

Numbering Some segments may start with a sequence of numbers, namely the section and subsection titles. This attribute captures the cardinality of that sequence. If a section starts with "1.4.5" the attribute value is 3.

Context Another important factor in the classification of a segment is context information which reflects the segments surrounding the segment being considered. This allows the segment's neighborhood to influence its classification. Here it was chosen the use of:

Surrounding text (non-segment divider) This attribute indicates the presence of surrounding text. This is useful to distinguish between bold-faced font that is used to give emphasis to a block of text and a bold-faced font that is used in a title of a section, for example.

Image exists (non-segment divider) This attribute indicates the existence of an image. It is used to distinguish a image caption from other headings.

Next segment Image exists (post-processing) The value of the "Image exists" attribute of the following segment. This was found to be useful in recognizing figure captions that sometimes occur before the image.

Class of previous segment (post-processing) This attribute contains the classification of the previous segment (this attribute is added during the training and classification stages).

Table exists (post-processing) The same as the "Image exists" attribute but for tables.

In the case of tables, as opposed to images, their content is recognized as blocks of text, making the use of a "Next segment Table exists" attribute misleading. This was observed in our experiments, where the addition of this attribute reduced the performance of the classifier and, as such, it was not included.

The segmentation algorithm has two steps. In the first step, the XHTML page is processed resulting in a list of segments. In the second step, the result list is cleaned-up and post-processing attributes are applied.

1. This step corresponds to a depth-first transversal of the XML DOM (Document Object Model) [45] tree that corresponds to the XHTML page, starting at the `<body>` node.

   During transversal, the set of attributes is updated when entering an XML node that matches a XHTML format tag, and restored when leaving a node that changed the attribute values. Notice that not all XHTML tags will cause an attribute to change, either because they convey the same value as a previous tag or because they are not used in any of the attributes.

   When visiting a XML node, one of two things can happen:

   (a) If it is a text node, then the text contained in it is collected by concatenation with a *current* text string;

   (b) If it is a format tag node and the attributes change, then the *current* text string is used to build a new segment. The new segment will have the attribute values

as they were prior to the update and the *current* string is reset to an empty string.

2. In this step the segment list is cleaned up by executing the following steps:

   (a) Merge consecutive segments in the list that share the same attribute values.

   (b) Remove segments that have only whitespace as text. Whitespace only segments are usually the result of the use of whitespace to promote page alignment. This step is not done simultaneously with the previous step because the "empty" segments removed in this step are usually text dividers and as such the segments they split should not be merged.

   (c) Apply the attributes that are derived from the segments (post-processing type).

   (d) Normalize attributes that require it. In this work the only attribute normalized is the font size, as previously explained.

3.1.2 Naive Bayes Classifier

In a Bayes classifier, an instance is described by a finite set of attribute values ($A_1 = v_{k1}, ..., A_n = v_{kn}$). There is a finite set C of source classes that might have generated the instance and the idea is to choose the correct one based on information provided by a labeled training set. In probabilistic terms, each new instance is assigned to the class that has the maximum *a posteriori* probability (MAP):

$$C_{MAP} = \underset{C_i \in C}{argmax}\, P(C_i | A_1 = v_{k1}, ..., A_n = v_{kn}) \tag{3.1}$$

Obtaining a probability estimation for this formulation is infeasible, however Eq. 3.1 can be rewritten using Bayes' theorem:

$$P(C_i | A_1 = v_{k1}, ..., A_n = v_{kn}) = \frac{P(A_1 = v_{k1}, ..., A_n = v_{kn} | C_i).P(C_i)}{P(A_1 = v_{k1}, ..., A_n = v_{kn})} \tag{3.2}$$

resulting in the following expression:

$$C_{MAP} = \underset{C_i \in C}{argmax}\, \frac{P(A_1 = v_{k1}, ..., A_n = v_{kn} | C_i).P(C_i)}{P(A_1 = v_{k1}, ..., A_n = v_{kn})}. \tag{3.3}$$

Furthermore, observing that, for a given instance, the denominator is constant, Eq. 3.3 is equivalent to:

$$C_{MAP} = \underset{C_i \in C}{argmax}\, P(A_1 = v_{k1}, ..., A_n = v_{kn} | C_i).P(C_i). \tag{3.4}$$

The Naive Bayes simplifying hypothesis is that the attribute values are conditionally independent given the target value. The resulting classifier is:

$$C_{NB} = \underset{C_i \in C}{argmax}\, P(C_i) \prod_j P(A_j = v_{kj} | C_i) \tag{3.5}$$

3.1.3 Segment Classification

As a result of the document segmentation stage each document can be seen as a string of segments, where each segment is associated with a set of attribute value pairs. What remains to complete the tokenization process is labeling each segment with a primitive concept, which is a classification problem.

With a choice between supervised and unsupervised classification methods, the choice was in favor of a supervised one. With a supervised classification method the set of primitive concepts and their names are determined by hand. This set of concepts will determine the alphabet used in the language learning part and should result in a cleaner start for the next steps, assuming that the choice of the set of primitive concepts is a good one, as it eliminates one possible source for errors. It also solves the problem of naming the primitive concepts, a chore that would have to be done by a person anyway.

The classifier chosen was the Naive Bayes classifier. The Naive Bayes classifier is robust in face of noise in the data and, besides providing for a simple implementation, usually has a comparable performance to other methods.

To use the Naive Bayes classifier the following probabilities were estimated during the training phase:

- $P(C_i)$, where C_i is one of the chosen classes.

$$P(C_i) = \frac{segments_i}{|examples|} \tag{3.6}$$

  where $segments_i$ is the number of segments classified in C_i and $|examples|$ is the number of segments presented in the training set.

- $P(A_j = v_k|C_i)$, where A_j is an attribute and v_k is one of its values. Here the *m-estimate* of probability is used:

$$P(A|C) = \frac{n_c + mp}{n + m} \tag{3.7}$$

  where n_c is the number of times A is classified in C, n is the total number of examples of A, p is the prior estimate of the probability we wish to determine and m is a constant called *equivalent sample size*, which determines how heavily to weight p relative to the observed data.

  The prior used was $p = \frac{1}{k}$ [62] , where k is the total number of different attribute-value pairs that where observed ($|Vocabulary|$) and $m = k$.

$$P(A_j = v_k|C_i) = \frac{n_l + 1}{n + |Vocabulary|} \tag{3.8}$$

Here n_l is the number of times $A_j = v_k$ is classified as C_i, n is the number of occurrences of $A_j = v_k$.

The classification phase is done using Eq. 3.5, but ignoring attribute-value pairs that have not been seen during the training phase.

3.1.4 Experimental Results

To evaluate the tokenization method, a set of 25 technical papers in HTML was collected from several World Wide Web Conferences[2]. The papers were segmented and the segments manually classified into the following classes (Author(s), Title, AbstractTitle, AbstractText, IndexTitle, IndexText, Figure(or Table)Caption, ConferenceTitle, SectionTitle, SubSectionTitle, SubSubSectionTitle, SimpleText, FormatedText). These classes represent the vast majority of the structure elements present in articles.

The keywords used were: "abstract", "figure", "table", "contents" and "overview".

Note that the attribute that indicates the classification of the previous segment is updated dynamically during the classification to reflect the result of the actual classification of the previous segment (having a value of "null" for the first segment of each document). During the training phase this attribute is extracted from the previous segment (having a value of "null" for the first segment of each document). The segmentation resulted in 5472 segments which were then used to perform a ten fold cross-validation. The results are presented in Table 3.1 and Table 3.2.

Table 3.1: Classification Error

Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	Average
7.68%	6.22%	2.38%	3.47%	1.83%	2.01%	0.73%	2.74%	0%	4.02%	3.11%

3.1.5 Discussion

Document tokenization, using the approach just described, resulted in a good overall classification error. Nevertheless, there are a few classes that have a high classification error. Some reasons for the high error rates appear to be the following.

The IndexTitle and IndexText classes have two problems. The IndexTitle is, in terms of text format, identical to a title, for example a section title. Moreover, it's position in the beginning of a document also causes it to be mistaken for an abstract title. The IndexText appears similar to a regular text segment and, when the IndexTitle is mistaken

[2]WWWC 96, WWWC 2000, WWWC 2001, WWWC 2002 and WWWC 2003

Table 3.2: Classification data – target vs. obtained

	CnfT	Ttl	Aths	AT	ATx	FTx	STx	ST	SST	IdxT	Idx	SSST	FigC	Error
CnfT	9	0	0	0	0	0	0	0	0	0	0	0	0	0%
Ttl	0	25	1	0	0	0	0	0	0	0	0	0	0	3.85%
Aths	0	0	18	0	0	5	5	4	0	0	0	0	0	43.75%
AT	0	0	0	22	0	1	0	0	0	0	0	0	0	4.35%
ATx	0	0	0	0	13	0	7	0	0	0	0	0	0	35%
FTx	0	0	9	2	3	2204	17	3	32	0	0	10	1	3.38%
STx	0	0	0	0	5	17	2511	0	1	0	0	2	0	0.99%
ST	0	0	0	0	0	0	0	196	3	0	0	0	0	1.51%
SST	0	0	0	0	0	4	0	9	179	0	0	0	0	6.77%
IdxT	0	0	0	2	0	0	0	1	0	0	0	0	0	100%
Idx	0	0	0	0	2	0	1	0	0	0	0	0	0	100%
SSST	0	0	0	0	0	2	0	0	12	0	0	35	1	30%
FigC	0	2	0	0	0	3	1	0	2	0	0	0	88	8.33%

ConferenceTitle – CnfT, Title – Ttl, Author(s) – Aths, AbstractTitle – AT, AbstractText – ATx, FormatedText – FTx, SimpleText – STx, SectionTitle – ST, SubSectionTitle – SST, IndexTitle – IdxT, IndexText – Idx, SubSubSectionTitle – SSST, FigureCaption – FigC

for an abstract title, it is confused with the abstract text, due to the use of the attribute indicating the previous classification.

The AbstractText class, like the IndexText class, is very similar to a regular text segment. Moreover, it is not always presented in a single segment, but may appear as several SimpleText and FormatedText segments.

The Author(s) class is difficult to classify because it has many different presentation styles. To correctly recover this information would require, in our opinion, a more detailed processing of the text, including the position in the page.

Finally, the SubSubSectionTitle, SubSectionTitle and SectionTitle classes are sometimes confused because of the varying font sizes that are used to present them. This is somehow compensated by the normalization of the font size attribute, but not completely. Additionally in some documents all section, subsection, etc. titles have the same font size.

3.2 Language Learning

This part of the learning process is concerned with the inference of a description of the implicit document structure by means of some language class. This will enable the next step of the process to infer concepts formed using the primitive concepts used in the tokenization step by means of capturing the regularities in the language description.

The task of learning a regular language can be seen as learning a DFA that accepts the strings that belong to the language.

In this work, language learning was restricted to the class of regular languages. The target application, technical papers, did not require a more expressive language class, as will, hopefully be evident from the results and using a more expressive language would unnecessarily augment the problem complexity.

As should be evident by the problem description, we are in a setting where the available learning material is not only scarce (in comparison with the amount of data typically available to machine learning methods for language learning), but it is also formed exclusively by positive samples.

Two distinct approaches to the problem were explored. The first was the use of active learning, as described in the next section, and the second was the application of the Minimum Description Length (MDL) principle to DFA learning, described in Section 3.2.2.

The first approach, while easier to use, requires, in our setting, a human teacher. This naturally may result in a mismatch between the inferred language and the actual document structure that will have to be dealt with when parsing for concept instances. The second approach, which does not require a teacher, suffers from the problem of resulting in an very large search space. This second approach could be considered an unsupervised learning method in the sense that there is no need for labeled samples as it uses only examples of the target concept.

3.2.1 Active Learning

The problem of learning a DFA can be approached in the active learning setting. In an active learning setting, the learning algorithm, referred to simply as learner, can pose questions about the target concept to an oracle, referred as the teacher.

In the case of regular languages, it has been shown [63] that for efficient learning it suffices to use only membership and equivalence queries.

membership queries the learner presents an instance for classification as to whether or not it belongs to the unknown language;

equivalence queries the learner presents a possible concept and the teacher either acknowledges that it is equivalent to the unknown language or returns an instance that distinguishes both.

Angluin presented an efficient algorithm [57] to identify regular languages from queries, the L* algorithm. This algorithm derives the minimum canonical DFA that is consistent with the answered queries.

In our setting, the use of the L* algorithm would require a human teacher, as we only have available a limited set of positive sample strings for the target language. However, the straight use of the L* algorithm, especially in the case of a human teacher, generates an impractical large number of queries. Therefore, a possible solution was developed to address this problem.

In the following, first the L* algorithm is presented and then the developed solution and results on the document structure inference problem are shown.

L* Algorithm

The L* algorithm implements a learner for regular languages, in the active learning setting. This learner requires a teacher that can answer membership and equivalence queries, referred to as a minimally adequate teacher.

In a DFA, if two states q_i and q_j are not equivalent, then there exists a string such that the state reached using that string from q_i and q_j will be of different type, one accepting state and the other not accepting. The L* algorithm uses this principle to distinguish the states in the DFA.

The strings used to distinguish states are called experiences and stored in a non-empty suffix-closed[3] set E. To identify states, a non-empty prefix-closed[3] set of strings S is maintained and each string $s \in S$ associated with a row of values $row(s)$, each value corresponding to a string in E. For a string $e \in E$, the value in $row(s)$ for e is 1 if $s \cdot e$ is accepted by the target DFA.

The L* algorithm maintains an observation table[4] $T : ((S \cup S \cdot \Sigma) \cdot E) \rightarrow \{0, 1\}$ where the rows are indexed by strings in $S \cup S \cdot \Sigma$ and the columns index by E. Row values are as described above and are obtained using membership queries. For example, in Table 3.3, $S = \{\lambda\}$, $E = \{\lambda\}$, $S \cdot \Sigma = \{0, 1, 2\}$ and no string with length ≤ 1 is accepted by the target DFA.

Table 3.3: Initial L* table at the first conjecture

		λ	} E
S {	λ	0	
$S \cdot \Sigma$	(2)	0	
	(1)	0	
	(0)	0	

[3]Suffix(prefix)-closed set is such that given an element in the set, every suffix(prefix) of that element is also in the set.

[4]Set concatenation $A \cdot B = \{ab | a \in A, b \in B\}$.

To represent a valid complete DFA, the observation table must meet two properties: closure and consistency. The observation table is *closed* iff, for each t in $S \cdot \Sigma$ there exists an $s \in S$ such that $row(t) = row(s)$. The observation table is *consistent* iff for every s_1 and s_2, elements of S, such that $row(s_1) = row(s_2)$ and for all $a \in \Sigma$, it holds that $row(s_1 \cdot a) = row(s_2 \cdot a)$.

The DFA $D = (Q, q_0, F, \delta)$ that corresponds to a *closed* and *consistent* observation table is defined by:

$Q = \{row(s) : s \in S\}$

$q_0 = row(\lambda)$

$F = \{row(s) : s \in S \wedge T(s) = 1\}$

$\delta(row(s), a) = row(s \cdot a)$

For example, Fig. 3.3 shows the DFA corresponding to the observation table shown in Table 3.4. Note that when a string belongs to both S and $S \cdot \Sigma$ then it is only represented once in the observation table (in the top part). This can be seen in Table 3.4. To obtain, for example, the transition function value for the initial state and symbol $0 \in \Sigma$, i.e. $\delta(row(\lambda), 0)$, one must lookup $row(\lambda \cdot 0)$, namely, the line labeled by (0). This line is shown in the top part of Table 3.4 because $(0) \in S$.

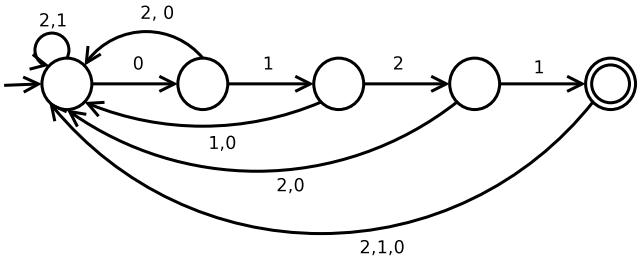

Figure 3.3: Intermediate DFA

It can be proved that any DFA consistent with the observation table, but different by more than an isomorphism, must have more states.

The L* algorithm, shown in Algorithm 1, starts with the initialization of the observation table using membership queries. Before a conjecture is made, the table is checked and updated until it is both *closed* and *consistent*.

If the observation table is not *consistent*, then there exist $s_1, s_2 \in S$, $a \in \Sigma$ and $e \in E$ such that $row(s_1) = row(s_2)$ and $T(s_1 \cdot a \cdot e) \neq T(s_2 \cdot a \cdot e)$. The set E is augmented with $a \cdot e$ and the observation table is extended using membership queries.

If the observation table is not *closed*, then there exist an $s_1 \in S$ and an $a \in \Sigma$ such that $row(s_1 \cdot a)$ is different from all $row(s)$ with $s \in S$. The set S is extended with $s_1 \cdot a$ and the observation table is extended using membership queries.

Table 3.4: L* table at the second conjecture

	λ	(1)	(2 1)	(1 2 1)
(0 1 2 1)	1	0	0	0
(0 1 2)	0	1	0	0
(0 1)	0	0	1	0
(0)	0	0	0	1
λ	0	0	0	0
(0 1 2 1 2)	0	0	0	0
(0 1 2 1 1)	0	0	0	0
(0 1 2 2)	0	0	0	0
(0 1 2 1 0)	0	0	0	0
(0 1 2 0)	0	0	0	0
(0 1 1)	0	0	0	0
(0 2)	0	0	0	0
(0 1 0)	0	0	0	0
(2)	0	0	0	0
(1)	0	0	0	0
(0 0)	0	0	0	0
(0 2 2 2)	0	0	0	0

Once the inner loop terminates, the DFA that corresponds to the observation table is presented to the teacher. If the teacher accepts this DFA the algorithm terminates. If the teacher returns a counter-example, then the counter-example and all of its prefixes are added to S, the table is extended using membership queries, and the algorithm continues with the first step (the loop that verifies the observation table properties).

For example, to learn the language $L = \{(0\ 1\ 2\ 1), (0\ 2\ 2\ 2)\}$, the algorithm starts a series of queries (shown in Table 3.5) until it reaches a closed and consistent observation table, Table 3.3. It then poses the first conjecture. The first conjecture is a DFA with only one state, that accepts no strings. After receiving the string (0 1 2 1) as (a negative) reply to the first equivalence conjecture, the algorithm poses a long sequence of membership queries (shown in Table 3.5) until it finally reaches a point where the observation table is again both *closed* and *consistent*, as shown in Table 3.4. Then the second conjecture, shown in Fig. 3.3, is presented to the teacher. The process could be continued until the DFA shown in Fig. 3.4 is reached[5] that accepts only the strings present in the target

[5]The resulting DFA would have an additional non-accepting state, here omitted for clarity, where all

Algorithm 1 L* Algorithm

```
S ← {λ}
E ← {λ}
T ← initialize observation table
loop
  while table not closed and consistent do
    update T using membership queries
  end while
  result ← equivalence query on conjecture(T)
  if result is OK then
    return conjecture(T)
  else if result is counter-example then
    update T with counter-example
  end if
end loop
```

language L.

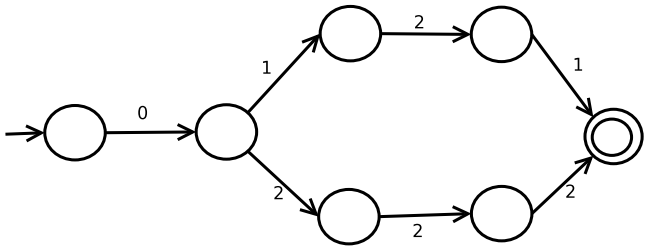

Figure 3.4: Example DFA (extra state removed)

A More Powerful Teacher

Query learning algorithms, such as L*, described in the previous section, produce a very large number of queries. This makes their use with human teachers impractical.

Depending on the target language, the number of queries of each type varies. Nevertheless, the number of membership queries is typically the dominant part in the total query count.

Membership queries have two possible answers, positive or negative. The negative answer is used to restrict the target language (with respect to the universal language containing all strings). As such, it is reasonable to expect that most languages will have a much larger count of negative membership queries than positive membership queries.

the transitions that are not shown in the figure would converge.

Table 3.5: L* execution trace

L*			L* with filter
λ ? N	(0 1 2 1 2) ? N	(0 2 2 1) ? N	λ ? N
(0) ? N	(0 1 2 1 2 1) ? N	(0 1 0 2 1) ? N	(0) ? (e (0))
(1) ? N	(0 1 2 1 1 1) ? N	(2 2 1) ? N	(1) ? (s (1))
(2) ? N	(0 1 2 2 1) ? N	(1 2 1) ? N	(2) ? (s (2))
	(0 1 2 1 0 1) ? N	(0 0 2 1) ? N	
(conjecture)	(0 1 2 0 1) ? N	(0 1 2 1 2 1 2 1) ? N	(conjecture)
	(0 1 1 1) ? N	(0 1 2 1 1 1 2 1) ? N	
(0 0) ? N	(0 2 1) ? N	(0 1 2 2 1 2 1) ? N	(0 1) ? (e (0 1))
(0 1) ? N	(0 1 0 1) ? N	(0 1 2 1 0 1 2 1) ? N	(0 2) ? (e (0 2))
(0 2) ? N	(2 1) ? N	(0 1 2 0 1 2 1) ? N	(0 1 1) ? (p ((1 1)))
(0 1 0) ? N	(1 1) ? N	(0 1 1 1 2 1) ? N	(0 1 2) ? (e (0 1 2))
(0 1 1) ? N	(0 0 1) ? N	(0 2 1 2 1) ? N	(0 1 2 1) ? Y
(0 1 2) ? N	(0 1 2 1 2 2 1) ? N	(0 1 0 1 2 1) ? N	(0 1 2 2) ? (p ((1 2) (2)))
(0 1 2 0) ? N	(0 1 2 1 1 2 1) ? N	(2 1 2 1) ? N	(0 2 1) ? (p ((0 2) (1)))
(0 1 2 1) ? Y	(0 1 2 2 2 1) ? N	(1 1 2 1) ? N	(0 1 0 1 2 1) ? (p ((0) (0)))
(0 1 2 2) ? N	(0 1 2 1 0 2 1) ? N	(0 0 1 2 1) ? N	
(0 1 2 1 0) ? N	(0 1 2 0 2 1) ? N		(conjecture)
(0 1 2 1 1) ? N	(0 1 1 2 1) ? N	(conjecture)	

To deal with the large number of membership queries, that typically happens when learning non-trivial automata, it is proposed, in this work, to use a more powerful teacher. Should the answer to a membership query be negative, the teacher is requested to return additional information, namely, to identify a set of strings that would result also in negative answers to membership queries.

Three forms were considered as possible answers:

1. A string prefix – This form identifies the set of strings that start with the same prefix and that are also negative examples. Its use can be seen in Table 3.5, with the form *(s <string>)*.

2. A string suffix – The second form does the same as the first one, but with the string suffix. It identifies strings that end in the same manner and that are also negative examples. Its use can also be seen in Table 3.5, with the form *(e <string>)*.

3. A list of substrings – The third form can be used to specify a broader family of strings

that are negative examples. Here one can identify strings by listing substrings that, when they are all present in a given string, in the same order, imply that the string is part of the described set and a negative example. For example, to identify the set of strings that contain two zeros, the reply would be the following list ((0)(0)), where (0) is a string with just one symbol, 0. Its use is also illustrated in Table 3.5, with the form *(p (<string1> ... <stringN>))*.

Note that all these specifications can be viewed as non-deterministic finite automata (NFA).

Using the additional information, the learner can now find out the answer to a number of membership queries without making an explicit query, simply by matching the strings with the stored information using the NFA corresponding to the new answer form. This can clearly be seen in Table 3.5, right hand side column, where the same DFA is inferred with and without the proposed extension, resulting in an important reduction in the number of queries.

Although this requires more sophisticated answers from the teacher, this is a reasonable request when dealing with human teachers. A human teacher must have some informal definition of the target language in his mind, to use a query learning approach, and it is reasonable to expect that most negative answers could be justified using the proposed method. The use of a query learning method when an informal definition is already present in the human teacher's mind is necessary to obtain a minimal DFA with less effort than it would require to manually build one. As such, the extra required effort is small, since the human teacher would already have identified that justification in order to answer the original membership query. Moreover, in a graphical environment this could be easily implemented by allowing for the selection of parts of the query string using a mouse pointer (allowing for multiple-selection to indicate a list of substrings answer).

The proposed solution uses the L* algorithm as a "black box". A filter is placed between the teacher and the learner, which records the additional information returned by the teacher on negative membership query answers. This information is then used to reply, whenever possible, to the learner without consulting the teacher.

Example Results and Equivalence Queries

Table 3.6 shows the results obtained for the example DFA from Fig. 3.4. In this example, the strings used to reply (negatively) to the equivalence queries were (0 1 2 1), (0 2 2 2) and (2 2 2 2). Table 3.7 shows the distribution of membership query answers by type and the number of answers made by the filter using the information of each type of query answer.

Even in this simple example, the number of membership queries is substantially reduced making the method usable by human teachers (L* with filter (A) in Table 3.6). Further results with a real application are shown in the next section.

Table 3.6: Query count results - simple example

	Membership Query Numbers		Equivalence Query Numbers
	Positive	Negative	
L*	2	185	4
L* with filter (A)	2	13	4
L* with filter (B)	2	13	3

(A) - Filter applied to membership queries;

(B) - Filter applied to membership and equivalence queries.

Table 3.7: Query counts by type - simple example

	Start with	End with	Has parts	Unjustified	Total
Teacher answers	2	4	5	2	13
Filter use counts	72	21	79	-	172

The extra information returned by the teacher could also be used to answer some equivalence queries, namely those containing strings that are not part of the target language and can be detected by the NFA already recorded. For example, the DFA in Fig. 3.3 admits strings, containing two or more 0s, that do not belong to the language. This could be detected as it was already stated in the end of Table 3.5 by "(p ((0) (0)))".

However, to detect these cases it would be necessary to obtain the product automaton between the recorded NFA and the DFA proposed by the learner, a process with quadratic complexity on the number of states. Note also that the number of states not only increases with the complexity of the language, but also with the number of extended answers (answers with the new proposed forms). This is a costly operation and would only remove, in general, a small fraction of the equivalence queries (L* with filter (B) in Table 3.6).

Results

The approach described was then applied to the problem at hand, the inference of document structure. For this problem, the alphabet used corresponds to the primitive concepts

already used in the tokenization step: ConferenceTitle (0), Title (1), Author (2), AbstractTitle (3), AbstractText (4), IndexTitle (6), IndexText (7), SectionTitle (8), SubSectionTitle (10), SubSubSectionTitle (11), SimpleText (5), FormatedText (9), FigureCaption (12). Furthermore, to apply this, it was assumed that a technical paper follows this set of rules:

- The ConferenceTitle is optional;
- There can be one or more Authors;
- The IndexText and IndexTitle are optional;
- A section can contain some of the text elements (SimpleText, FormatedText, FigureCaption) and lower level sections.

The equivalence queries were answered using the strings in Table 3.8.

Table 3.8: Strings used in equivalence queries

Q	Strings supplied to the algorithm as counter-examples
1	Title Author AbstractTitle AbstractText SectionTitle SimpleText
2	Title Title Author AbstractTitle AbstractText SectionTitle SimpleText
3	Title Author AbstractTitle AbstractText IndexTitle IndexText SectionTitle SimpleText
4	ConferenceTitle ConferenceTitle Title Author AbstractTitle AbstractText SectionTitle SimpleText
5	Title Author AbstractTitle AbstractText SectionTitle SimpleText SubSectionTitle SimpleText SubSubSectionTitle SimpleText

Table 3.9 shows the number of queries resulting from the use of the L* algorithm and of the proposed solution. The resulting DFA is shown in Fig. 3.5. Table 3.10 shows the distribution of membership query answers by type and the number of answers made by the filter using the information of each type of query answer.

As the results show, the number of negative membership queries is substantially reduced. Also, at least in this example, the negative membership queries are the largest in number. This is the case for automata that have few terminal states, relatively to the total number of states, a situation that is common in real cases.

As mentioned in Section 3.2.1, the information can be used to reduce the amount of equivalence queries (L* filter (B) in Table 3.9), but results in only a small reduction in the number of queries.

Table 3.9: Query count results

	Membership Query Numbers		Equivalence Query Numbers
	Positive	Negative	
L*	99	4118	6
L* with filter (A)	99	110	6
L* with filter (B)	99	110	5

(A) - Filter applied to membership queries;

(B) - Filter applied to membership and equivalence queries.

Table 3.10: Query counts by type

	Start with	End with	Has parts	Unjustified	Total
Teacher answers	11	10	88	1	110
Filter use counts	101	280	3627	-	4008

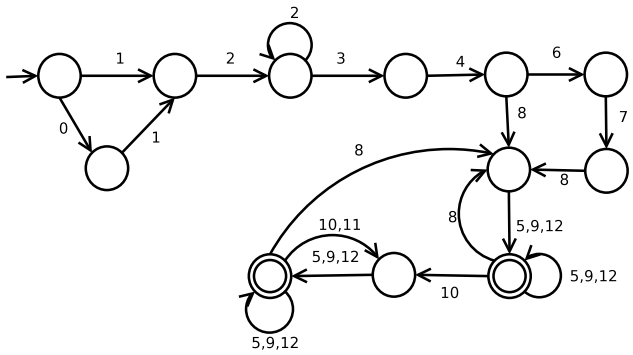

Figure 3.5: DFA representing the article structure (extra state removed)

Discussion

The solution presented in the this section, allows for a significant reduction in the number of membership queries. Moreover, the same approach can be used as a "black box" with other learner algorithms.

However, the use of this type of algorithms with a human teacher still leaves a problem, that of presenting a conjecture and recording a counter-example. Depending on the objective at hand, an appropriate transformation would have to be made to the conjecture for it to be presented in more familiar terms.

For the problem at hand, the inference of an ontology, if the assumption can be made that the user of such a method understands some formal representation of the ontology,

then that could be used to present the conjecture. However, care must be taken with such a course of action, for the transformation may mask the flaws in the conjecture.

The next step in the ontology building method being described provides for such a transformation, from DFA to an ontology, and could perhaps be used to provide a better interface to answer membership queries.

In the following section, however, an alternative is presented to the use of an active learning approach to this problem.

3.2.2 DFA Inference Using the Minimum Description Length Principle

This section deals with the problem of the inference of a Deterministic Finite Automaton (DFA) from a sample of the language that it accepts. Given a positive sample, the range of plausible languages is infinite, since it varies from the very specific language that contains only the strings present in the sample, to the opposite side of the spectrum where the language is the set Σ^*.

Given this inference problem, one way to approach it (the one used in this text) is to find some form of bias with which a choice can be made from the infinitely many available options.

The bias chosen is the use of the Minimal Description Length (MDL) [64–66] principle. This principle states, in rough terms, that one should "choose" the most compact description of the sample, where such description takes into account not only the encoding of the samples, but also the encoding of the choice of the "model" used to encode the sample. Using such an approach, one reaches a balance between "model" complexity and the "model" fit to the sample. A complex "model" will have a good fit to the sample, resulting in a reduced description length of the sample, but a big description length of the "model", and vice-versa. Theoretical and empirical analysis support the validity of this model, that has been applied successfully many times in the past [67–69].

There are several possible ways to perform such a description length measurement, one such way is using the Kolmogorov complexity, where the description length is the length of a program that produces the sample as its output.

A general, simple way, to approach the problem is with what is known as the two-part code approach. In a two-part code, the description length is the sum of the bit length of the hypothesis $L(H)$ with the bit length of the sample D encoded using the hypothesis $L(D|H)$. In our setting, the hypothesis would be a DFA and some scheme would have to be devised to obtain an encoding for the DFA and for the sample. Note that the "model" is regarded as a particular hypothesis.

There is a well known result from information theory that given a set of possible events and a probability distribution P over them, the optimal length of the bit code for each event x is given by $-\log P(x)$. If the "model" or hypothesis is a probability distribution over all possible sample elements, then we can take $L(D|H)$ as $-\log P(D|H)$. However, this still leaves us with the problem of encoding the hypothesis.

A possible approach to this problem is to organize the hypothesis into models, that describe them. Thus instead of associating an encoding for the sample D with a particular hypothesis we associate it with the model $\mathcal{H}$ that contains the hypothesis. The problem is then to find the model that minimizes $L(D|\mathcal{H})$. If the hypothesis H is represented by a probability distribution then a model is a probabilistic model $\mathcal{H}$ with some set of parameters. The code for $L(D|\mathcal{H})$ should be such that whenever there is a member of $\mathcal{H}$ that fits the data well, in the sense that $L(D|H)$ is small, then $L(D|\mathcal{H})$ is also small. Codes with this property are called universal codes. In fact, a two-part code can be used as an universal code. There are several universal codes. Among them the Normalized Maximum Likelihood is important because it can be shown to be optimal in the sense that the worst-case regret for any distribution is larger or equal to the regret of the Normalized Maximum Likelihood distribution, if the parametric complexity of the model is finite [70]. In the following, the Normalized Maximum Likelihood will be used as an universal code. A detailed description of this theory can be found in [66].

In the following, an adaptation of the MDL principle to DFA inference is described along with the resulting inference solution, some results and a discussion of this work.

MDL for DFA Inference

To infer a DFA, according to what was described above, it is necessary to define a probability distribution over the set of string samples with a predefined size and a way to organize them into probabilistic models.

First, an inference scheme will be devised without considering which states are final states. This information will be included later in an augmented scheme.

In the following, χ will be used to represent the set of possible items in the sample. In our case, $\chi = \Sigma^*$, the set of all strings. χ^n will represent the set of all possible samples of size n, where a particular sample will be represented by $x^n = (x_1, ..., x_n)$ and $x_i \in \chi$ is a string.

We will work with incompletely specified DFAs, that is, with DFAs for which not all state transitions are defined. Moreover, all defined states are considered as accepting states during the inference process. These will be called reduced DFAs, rDFAs for short, and are defined as follows.

Definition 1. ***rDFA*** $= (Q, \Sigma, \delta, q_0)$ *where* Q, Σ *and* q_0 *are defined as usual and where* $\delta : Q \times \Sigma \rightarrow Q \cup \{\oslash\}$ *such that there is a path between the initial state* q_0 *and any of the states in* Q.

Given an rDFA, inferred by the process that we will describe, a fully specified DFA is obtained by marking as accepting states only the states reached by the sample strings and creating a new non-accepting state to be the target of all undefined transitions.

The next step in adapting the MDL principle to DFA inference is to choose what are going to be the "models" and what is the probability of a string, given the model, so as to be possible to apply the Normalized Maximum Likelihood to our problem.

First we will define what is the probability of a string, given a rDFA, and will attempt to do this in a straight forward manner.

Let x_i be a sample string and H a rDFA:

Definition 2. $\mathbf{P(x_i|H)}$

Let $d : Q \times \Sigma \rightarrow \{0, 1\}$ *be such that* $d(q, a) = 1$ *iff* $\delta(q, a) \neq \oslash$ *and* $d(q, a) = 0$ *otherwise.*

Let also $p : Q \times \Sigma \rightarrow [0, 1]$ *be defined as*

$$p(q, a) = \frac{d(q, a)}{\sum_{b \in \Sigma} d(q, b)}.$$

Now we can extend the function δ*, such that* $\delta' : Q \cup \{\oslash\} \times \Sigma \rightarrow Q \cup \{\oslash\}$ *is identical to* δ *if the first argument is a state in* Q *and takes on the value* $\oslash$ *when the first argument is* $\oslash$.

Given this,

$$P(x_i|H) = P(x_{i_0}...x_{i_k}|H) = p(q_0, x_{i_0}) \times p(\delta'(q_0, x_{i_0}), x_{i_1}) \times ... \times p(\delta'(\delta'(...), x_{i_{k-1}}), x_{i_k})$$

if x_i *is not the empty string. If* x_i *is the empty string,* $P(x_i|H) = 1$.

This does not correctly define a probability distribution because the sum of $P(x_i|H)$ over χ is not one. Moreover, it is not even normalizable because the sum $P(x_i|H)$ over χ does not converge. However, this can be fixed by introducing a new constant α such that $0 < \alpha < \frac{1}{S}$, where S is the number of symbols in Σ, and changing the definition of $p : Q \times \Sigma \rightarrow [0, 1]$ to:

$$p(q, a) = \alpha \frac{d(q, a)}{\sum_{b \in \Sigma} d(q, b)}$$

Using this new version of $p(q, a)$, it follows that $0 \leq P(x_i|H) \leq \alpha^k$ where $k = length(x_i)$ and so:

$$\sum_{x_i \in \chi} P(x_i|H) < 1 + \sum_{k=1}^{\infty} (\alpha S)^k = 1 + \frac{\alpha S}{1 - \alpha S} = \frac{1}{1 - \alpha S}$$

that is finite because $\alpha S < 1$.

Now we can normalize $P(x_i|H)$, as:

$$P_{norm}(x_i|H) = \frac{P(x_i|H)}{\frac{1}{1-\alpha S}} = P(x_i|H)(1-\alpha S)$$

We can extend this definition to a sample $x^n = (x_1, ..., x_n)$ as:

$$P_{norm}(x^n|H) = \prod_{i=1}^{n} P_{norm}(x_i|H)$$

Having defined how to use a rDFA as a probability distribution, it remains to define what a "model" is. A probabilistic model implies the existence of a set of probabilistic distributions with similar characteristics and a fixed set of parameters with which to choose a particular probability distribution of that model. In this particular case we also want our set of models to have a growing complexity, for some measurement of complexity. Two ways to organize rDFAs into models are to group them either by the number of states or by the number of defined transitions. We choose to group them by the number of defined transitions. Using this grouping, the model parameters will be the particular way to organize the transitions, that is, for each transition, what is its start state, end state and label. Note that not all choices produce a correct (in the sense of connected and deterministic) rDFA. From now on we use $\mathcal{M}^j$ to represent the model given by the set of rDFAs with j defined transitions.

We now have the needed definitions to use the Normalized Maximum Likelihood to produce a MDL formulation. We will use $\hat{\theta}^{(j)}(x^n)$ to refer to the maximum likelihood estimation for the parameters of $\mathcal{M}^j$. Given x^n, $\hat{\theta}^{(j)}(x^n)$ is the rDFA in $\mathcal{M}^j$ that gives the maximum probability to x^n. The Normalized Maximum Likelihood for our probability distribution is:

$$P_{nml}(x^n|\mathcal{M}^j) = \frac{P_{norm}(x^n|\hat{\theta}^{(j)}(x^n))}{\sum_{y^n \in \chi^n} P(y^n|\hat{\theta}^{(j)}(x^n))}$$

Using a well known result from information theory that relates the bit length of the description of an event with the probability of that event, description length of x_n is:

$$-\log P_{nml}(x^n|\mathcal{M}^j) = -\log P_{norm}(x^n|\hat{\theta}^{(j)}(x^n)) + \log \sum_{y^n \in \chi^n} P(y^n|\hat{\theta}^{(j)}(x^n))$$

The second term in the right hand side of this expression is the parametric model complexity (from here on referred simply as model complexity), and is written as:

$$\mathbf{COMP}_n(\mathcal{M}^j) = \log \sum_{y^n \in \chi^n} P(y^n|\hat{\theta}^{(j)}(x^n))$$

This formulation only makes sense if $\mathbf{COMP}_n(\mathcal{M}^j)$ is finite, which we will prove next.

COMP$_n(\mathcal{M}^j)$ *is finite.* **Proof:** For that we will show that $\mathbf{COMP}_n(\mathcal{M}) > \mathbf{COMP}_n(\mathcal{M}^j)$ is finite, that is, we will prove that the model formed by all rDFAs has a finite complexity, and from this it follows that any subset of such a model will also have a finite complexity.

Using $k = \sum_{j=1}^{n} length(x_j)$ it follows from the definition of $P_{norm}(x^n|H)$ that

$$P_{norm}(x^n|\hat{\theta}(x^n)) \leq (1-\alpha S)^n \alpha^k,$$

and that:

$$\sum_{x^n \in \chi^n} P_{norm}(x^n|\hat{\theta}(x^n)) < \sum_{k=0}^{\infty} \left[S^k f_n(k) \alpha^k (1-\alpha S)^n\right] = (1-\alpha S)^n \sum_{k=0}^{\infty} \left[f_n(k)(\alpha S)^k\right]$$

where,

$$f_n(k) = \begin{cases} 1 & \text{para } k = 0 \\ \sum_{i=0}^{k-1} \binom{k-1}{i}\binom{n}{i+1} & \text{para } 0 < k < n \\ \sum_{i=0}^{n-1} \binom{k-1}{i}\binom{n}{i+1} & \text{para } k \geq n \end{cases}$$

To show that the right hand side of the inequality is finite, it suffices to show that $\sum_{k=0}^{\infty} \left[f_n(k)(\alpha S)^k\right]$ converges. Let $\beta = \alpha S$, $a_k = f_n(k)$ and $b_k = \beta^k$. To satisfy the D'Alembert convergence criterion, it suffices to show that, beyond some k, it holds that:

$$\lim_{k\to\infty} \frac{a_{k+1}b_{k+1}}{a_k b_k} = \lim_{k\to\infty} \frac{f_n(k+1)}{f_n(k)}\beta < 1$$

With $k \geq n$, we have:

$$\lim_{k\to\infty} \frac{f_n(k+1)}{f_n(k)}\beta = \beta \lim_{k\to\infty} \frac{\sum_{i=0}^{n-1}\binom{k}{i}\binom{n}{i+1}}{\sum_{i=0}^{n-1}\binom{k-1}{i}\binom{n}{i+1}} = \beta \lim_{k\to\infty} \frac{\sum_{i=0}^{n-1}\frac{k}{k-i}\binom{k-1}{i}\binom{n}{i+1}}{\sum_{i=0}^{n-1}\binom{k-1}{i}\binom{n}{i+1}}$$

As $i \leq n-1$ and n is a constant, when $k \to \infty$, we have that $\frac{k}{k-i} \to 1$, and it follows that:

$$\beta \lim_{k\to\infty} \frac{\sum_{i=0}^{n-1}\frac{k}{k-i}\binom{k-1}{i}\binom{n}{i+1}}{\sum_{i=0}^{n-1}\binom{k-1}{i}\binom{n}{i+1}} = \beta$$

that is,

$$\lim_{k\to\infty} \frac{a_{k+1}b_{k+1}}{a_k b_k} = \beta < 1$$

as we wanted to prove. □

Despite our efforts we were unable to come up with an exact formula to obtain the model complexity (or a close upper bound) for a fixed number of transitions, and therefore we were forced to come up with an alternative formulation. Our alternative formulation is closer to the two-part code. We count the number of rDFAs for a given number of transitions and use the number of bits required for that encoding as the rDFA complexity. Note that all rDFAs that are in the same model $\mathcal{M}^j$ will have the same complexity, as before. We maintain the same formulation for the sample complexity.

Our rough estimation of the number of rDFAs, given j transitions, is as follows. For j transitions, we can build rDFAs with a number of states from $\lceil \frac{j}{S} \rceil$ to $j+1$. As the rDFAs must be connected, then there must exist a tree using some portion of the j transitions that links all of the states to the start state. Now, using the *Prüfer* code, there exist, for z nodes, z^{z-2} trees. Note that here we have the first estimation error, since this code counts trees with named nodes, and as such there are several duplicates. These trees use $z-1$ transitions.

Leaving aside that not all ways to label the transitions are valid, we have the following estimate for the number of rDFAs with j transitions:

$$count(j) = S^j \left[\sum_{i=\lceil \frac{j}{S} \rceil}^{j+1} (i^{i-2})(i^{2(j-(i-1))}) \right] = S^j \left[\sum_{i=\lceil \frac{j}{S} \rceil}^{j+1} i^{2j-i} \right]$$

and so, an estimate for the complexity is:

$$\widehat{\mathbf{COMP}}_n(\mathcal{M}^j) = \log count(j) = j \log S + \log \left[\sum_{i=\lceil \frac{j}{S} \rceil}^{j+1} i^{2j-i} \right]$$

To choose the rDFA for a sample x^n, we must find the j and $\hat{\theta}^{(j)}$ that minimize:

$$\begin{aligned} MDL(x^n|j) &= -\log P_{norm}(x^n|\hat{\theta}^{(j)}) + \widehat{\mathbf{COMP}}_n(\mathcal{M}^j) \\ &= -\log \left[(1-\alpha S)^n P(x^n|\hat{\theta}^{(j)}(x^n)) \right] + \widehat{\mathbf{COMP}}_n(\mathcal{M}^j) \\ &= -\log \left[(1-\alpha S)^n \alpha^{\sum_{i=1}^n length(x_i)} P'(x^n|\hat{\theta}^{(j)}(x^n)) \right] + \widehat{\mathbf{COMP}}_n(\mathcal{M}^j) \\ &= C_1 - \log P'(x^n|\hat{\theta}^{(j)}) + \widehat{\mathbf{COMP}}_n(\mathcal{M}^j) \end{aligned}$$

where, $P'(x^n|H)$ is the original definition of P and

$$C_1 = -n \log(1-\alpha S) - \left[\sum_{i=1}^{n} length(x_i) \right] \log \alpha$$

Adding Final State Information

The inference scheme just described doesn't make any distinction between DFAs whose states have all transitions defined. Therefore, for languages that are represented by DFAs in such conditions, the inferred DFA is the single-state DFA that accepts all strings. To make it possible to distinguish between such languages, information about final states must be considered during inference. Note however that this will be done without the need for negative sample strings.

As the described inference scheme is based on transition costs, the cleanest way to add information regarding final states is to define them as special transitions that don't

consume alphabet symbols (a sort of epsilon transitions). We will consider a slightly different form of DFA, lets call it sfDFA, where there exists only one final state. Moreover, this final state can only be reached using our new special empty transitions. A DFA can be transformed into a sfDFA simply by adding the new state, the final state, and for each old final state adding an empty transition to the new final state, as shown in figure 3.6.

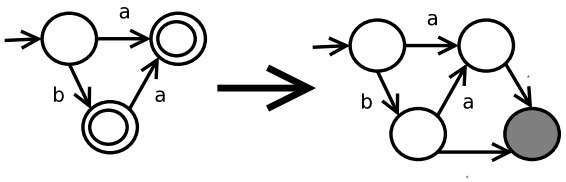

Figure 3.6: Encoding final states as a special transition

Transition Cost To account for the new transitions, a change has to be made to the definition of $P(x_i|H)$, by changing the transition cost. In the previous scheme, the transition cost was given by:

$$p(q,a) = \alpha \frac{d(q,a)}{\sum_{b\in\Sigma} d(q,b)},$$

with the new transitions, there are two cases:

1. The cost of a regular transition:

$$p(q,a) = \alpha \frac{d(q,a)}{final(q) + \sum_{b\in\Sigma} d(q,b)},$$

   where $final(q) = 1$ if q is a final state and zero otherwise.

2. The cost of accepting the string, that is, taking a special transition to the end state:

$$p(q, \text{ACCEPT}) = \alpha \frac{final(q)}{final(q) + \sum_{b\in\Sigma} d(q,b)}$$

This change has limited consequences in the results shown for the previous inference scheme. As there is a new transition required to accept a given string, it follows that $0 \leq P(x_i|H) \leq \alpha^{k+1}$ and

$$P_{norm}(x_i|H) = \frac{P(x_i|H)(1-\alpha S)}{\alpha}.$$

This adds a constant factor of $\frac{1}{\alpha^n}$ to the upper limit of $\mathbf{COMP}_n(\mathcal{M})$.

Model Complexity The change to the transition cost and the addition of the information about the final states has an affect on the model complexity. There are two possible ways to incorporate this into the inference scheme:

1. The first way is to not consider the special transitions as regular transitions and to account for the final state information as a part of the model parameters, $\theta^{(j)}$. This implies a change to the model complexity estimate, $\widehat{\mathbf{COMP}}_n(\mathcal{M}^j)$, accounting for the extra dimension:

$$\widehat{\mathbf{COMP}}_n(\mathcal{M}^j) = j \log S + \log \left[\sum_{i=\lceil \frac{j}{S} \rceil}^{j+1} i^{2j-i} 2^i \right]$$

2. A second approach consists of simply treating the special transitions as regular transitions and as such keeping the existing complexity estimate, but with an increased number of transitions depending on the particular sfDFA under consideration.

The second approach is not very coherent with the inference scheme that was described, as it makes an additional split of the model $\mathcal{M}^j$ into several models $\mathcal{M}^{j,t}$, where t is the number of special transitions.

With the first approach, to choose an DFA for a sample x^n, we must find the j and $\hat{\theta}^{(j)}$ that minimizes:

$$\begin{aligned}
MDL(x^n|j) &= -\log P_{norm}(x^n|\hat{\theta}^{(j)}) + \widehat{\mathbf{COMP}}_n(\mathcal{M}^j) \\
&= -\log \left[\frac{(1-\alpha S)^n}{\alpha^n} P(x^n|\hat{\theta}^{(j)}(x^n)) \right] + \widehat{\mathbf{COMP}}_n(\mathcal{M}^j) \\
&= -\log \left[\frac{(1-\alpha S)^n}{\alpha^n} \alpha^{\sum_{i=1}^n (length(x_i)+1)} P'(x^n|\hat{\theta}^{(j)}(x^n)) \right] + \widehat{\mathbf{COMP}}_n(\mathcal{M}^j) \\
&= C_1 - \log P'(x^n|\hat{\theta}^{(j)}) + \widehat{\mathbf{COMP}}_n(\mathcal{M}^j)
\end{aligned}$$

where, $P'(x^n|H)$ is identical to P but without the α factor at each transition, and

$$C_1 = -n \log(1-\alpha S) - \left[\sum_{i=1}^n length(x_i) \right] \log \alpha$$

Inference Solution

Given the formulation of MDL done in the previous section, there remains the problem of actually finding the j and $\hat{\theta}^{(j)}$ that minimize $MDL(x^n|j)$ and by that finding a DFA.

First, and to allow for comparison with heuristic search, a full search algorithm was implemented. The algorithm uses a depth first search with backtracking, where at each expansion an edge is added either between two states already present in the current

solution or between one of these states and a new state to be added. Not all new edges are considered. Only edges that increase the sample cover (the number of symbols of the sample strings that can be processed until a missing transition is reached), are considered until a full cover is reached. Note that this can only eliminate suboptimal solutions. The search is done with a limited depth, retrieving all optimal solutions with depth less than or equal to the specified depth. To obtain all solutions, one can use the PTA edge count as a limit depth. Also note that this search applies the first formulation of the inference scheme and doesn't account for final state information, as this additional information increases the search space.

The full search approach was, however, quickly abandoned, as it was only useful for very small samples, that resulted in a very small number of defined transitions in the PTA.

State merging [71] has allowed DFA inference to advance to large sized problems that would otherwise be intractable with simple search. In this approach, common to many DFA inference algorithms, the search starts with a prefix-tree automaton built from the labeled examples and successive state merges are performed until some condition is met. In the setting of this work, the labeled examples only contain positive strings and the target is the DFA that minimizes the MDL value.

The state-merging approach guarantees that the all of the DFAs encountered accept the training set. However, contrary to the situation were negative labeled strings are available, all state merges are valid, resulting in a larger search space than it is common in other state-merging applications.

After experimenting with a number of possibilities, it was chosen to use two search procedures, a greedy search and an extended form where greedy search is performed on all leafs of a depth limited exhaustive search from the the initial DFA. In both cases, the MDL value is used to guide the search and the state merge is performed using the standard merge algorithm [71], with a step that recursively merges states until a deterministic automaton is obtained. A node is not expanded if it's MDL value is larger than it's predecessor.

To reduce the search space, a hash function is used to keep track of previously explored nodes.

Results

To test this approach, a set of synthetic DFAs was used. These can be found in Fig. 3.7 to Fig. 3.10.

To generate the sets of training strings for each DFA, all strings up to a certain size were generated and, for each string accepted by the DFA, it was added to the training set

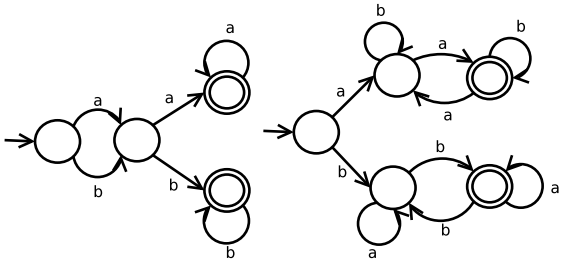

Figure 3.7: DFA 1 e 2

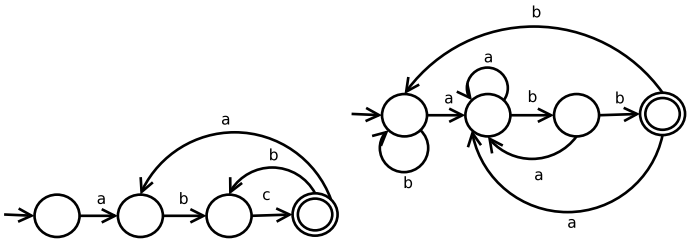

Figure 3.8: DFA 3 e 4

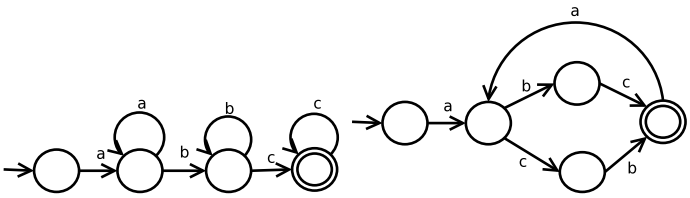

Figure 3.9: DFA 5 e 6

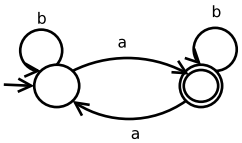

Figure 3.10: DFA 7

with a probability of 0.8. Sampling was done from the smallest strings and then growing the string size until the sample set reached the desired size.

Tests were done for both variations of the encoding of accepted states. When the pure greedy search did not work, a one-step deep depth-limited search followed by a greedy search from the leafs was used. Table 3.11 shows the results for the first approach (encode the end state information in the model parameters) and Table 3.12 shows the results for the second approach.

Table 3.11: Results for the first approach

DFA	Sample Size	$\sum_i^n length(x_i)$	Greedy Search Sufficient	Generated	MDL value
1	20	107	Yes	778	156.07
2	-	-	-	-	-
3	10	76	Yes	386	71.92
4	-	-	-	-	-
5	10	49	Yes	270	81.59
6	10	69	Yes	509	78.59
7	20	73	No	249216	147.18

Table 3.12: Results for the second approach

DFA	Sample Size	$\sum_i^n length(x_i)$	Greedy Search Sufficient	Generated	MDL value
1	20	107	No	320913	162.49
2	-	-	-	-	-
3	10	76	Yes	386	73.27
4	-	-	-	-	-
5	10	49	Yes	270	82.60
6	10	69	Yes	509	79.60
7	20	73	No	249885	148.24

The results point to the use of the first alternative for the encoding of accepting states in model complexity, that encodes then as model parameters.

To further test the inference using the first model complexity hypothesis, it was applied to the Tomita grammar set[72]. The Tomita grammar set is composed by the following grammars:

1. 1^*
2. $(10)^*$
3. no odd-length 0-string anywhere after an odd-length 1-string
4. no more than two 0's in a row
5. even length, such that $\#01's + \#10's = 0 \, mod \, 2$
6. $abs(\#1's - \#0's) = 0 \, mod \, 3$
7. $0^*1^*0^*1^*$
8. 0^*1
9. $(0^* + 2^*)1$
10. $(00)^*(111)^*$
11. even number of 0's and odd number of 1's
12. $0(00)^*1$
13. even number of 0's
14. $(00)^*10^*$
15. $12^*1 + 02^*0$

To perform the tests, a sample was generated for each language using a simple algorithm that generates strings following a distribution consistent with the one assumed in the computation of the description length. To generate a string from the DFA, the algorithm starts at the initial state and selects one of the available choices, with equal probability. A choice is either to follow one of the defined transitions or, if the state is an accepting state, to terminate the string. This generation model follows closely the proposed MDL definition and gives a probability distribution over the accepted strings that decreases exponentially with the string length.

The actual sample size was chosen to be five times the sample size that leads to an MDL value of the target DFA being approximately equal to the MDL value of the (single state) DFA that accepts all strings. Random samples of increasingly larger sizes were generated until this criterion was met. The idea is to generate a sample that is large enough to have enough information to distinguish the target DFA from the other possibilities. The resulting sample sizes are shown in Table 3.13.

Table 3.13: Sample information

Lang. #	n	$\sum_i^n length(x_i)$	$\|\Sigma\|$
1	5	16	2
2	5	32	2
3	60	533	2
4	225	2155	2
5	30	456	2
6	35	448	2
7	120	1016	2
8	5	17	2
9	20	135	3
10	10	124	2
11	30	448	2
12	10	122	2
13	5	32	2
14	15	232	2
15	20	135	3

In Table 3.14, we can see the inference results, particularly the expected and obtained MDL value. This value is identical when the obtained solution is the target DFA or an equivalent DFA. The results were obtained on a Pentium IV 3 GHz machine and the time represents the process user time. In the table, "Search type" corresponds to the depth of the exhaustive part of the search; a value of zero corresponds to a greedy search, a value of one to a greedy search on all descendents of the PTA.

One immediate conclusion that follows from both the generated sample sizes and from the results is that the method works better on sparse DFAs. The method was able to find the target DFAs in twelve out of fifteen languages. However, we can also see that some DFAs were not found, for languages 6, 5 and 11, obtaining instead over-generalized DFAs. This happens even though the target DFA has smaller MDL value than the one found by the algorithm, and suggests that the MDL value may not be a good heuristic to guide the search in all situations. For language 6, the result obtained was the DFA that accepts all strings and for languages 5 and 11, the result obtained was the DFA that accepts an even number of 1s and the DFA that accepts an odd number of 1s, respectively. Also notice that for language 7, the more extensive search found a slightly better solution, that captured a regularity in the sample, namely that a lot of strings did not have leading

Table 3.14: Inference results

Lang. #	Search type	Generated DFAs	MDL		# edges	Time (sec.)
			expected	obtained		
1	0	29	**24.58**	**24.58**	1	0
	1	133	**24.58**	**24.58**	1	0
2	0	212	**28.39**	**28.39**	2	0
	1	3112	**28.39**	**28.39**	2	0.03
3	0	49508	853.54	947.28	2	1.26
	1	43021301	**853.54**	**853.54**	5	1176.13
4	0	311993	3695.79	3779.60	2	1.87
	1	1670849567	**3695.79**	**3695.79**	5	120222.7
5	0	107388	614.61	777.68	2	3.39
	1	36289267	614.61	643.14	4	1261.58
6	0	84541	615.03	772.93	2	2.44
	1	40734974	615.03	772.93	2	1276.61
7	0	31332	**1542.29**	**1542.29**	7	0.96
	1	41336039	1542.29	1485.54	9	1365.94
8	0	68	**24.39**	**24.39**	2	0
	1	311	**24.39**	**24.39**	2	0.01
9	0	1278	**167.59**	**167.59**	6	0.02
	1	46270	**167.59**	**167.59**	6	0.5
10	0	1930	**105.15**	**105.15**	6	0.02
	1	362676	**105.15**	**105.15**	6	2.69
11	0	97088	595.50	765.00	2	1.98
	1	39401994	595.50	621.68	4	844.89
12	0	565	**73.07**	**73.07**	3	0.01
	1	39605	**73.07**	**73.07**	3	1.56
13	0	107	**28.39**	**28.39**	2	0
	1	2276	**28.39**	**28.39**	2	0.04
14	0	1295	**159.33**	**159.33**	4	0.01
	1	256868	**159.33**	**159.33**	4	4.82
15	0	1125	**167.59**	**167.59**	6	0.01
	1	81184	**167.59**	**167.59**	6	1.04

zeros, while preserving the language recognized by the DFA.

Discussion

The estimate used differs from the original probabilistic formulation in one, perhaps significant, way. It does not take into account the number of strings in the sample. Note that usually this is what is required of a two-part code, but it contradicts the probabilistic formulation where the sample size makes a difference in calculating the complexity of the model.

Although the MDL formulation makes sense from an inference standpoint, the model complexity estimate is probably too rough to work correctly with smaller sized samples. As the sample size grows, the value of the expected solution becomes better with respect to the remainder of the search space. This is a good indication that the problem is a defective prior, that is, an incorrect model complexity estimation.

Better results could come from either the combination of the MDL minimization with some other heuristic, or from a better search method, since in some cases the intended DFA does have in fact a lower value than the one found by the search.

The current search approach, although not dependent on the alphabet size, is dependent on the total size of the input (sum of the length of the strings) as, contrary to the usual application of state-merging, all merges are inherently valid.

3.3 Ontology Building

Having captured the document structure in a DFA, the next step, and the objective of the work presented in this section, is to transform that information into an ontology. What is intended is to capture the regularities in the source language and use them as the concepts in the ontology produced by this process. At the beginning of the process, the ontology only contains the primitive concepts, chosen during tokenization. Each new concept added to the ontology is then defined by means of relations to concepts already in the ontology.

One way to approach this problem would be to transform the DFA into a regular expression using a standard algorithm, such as the one shown in [51]. Then a mapping would be produced between blocks of the regular expression and concepts. However this approach has a problem: the regular expression produced by the standard algorithm is very long, unless extensive simplification is applied to the resulting regular expression.

The approach followed in this work is to recursively identify and replace local patterns in the DFA until a simple DFA, with one or two states and a single transition, is reached. Each identified local pattern, from a pre-defined set of pattern templates, is mapped to a

new concept and the ontology graph is progressively built up from the primitive concepts and the new concepts found during the process. It also provides a way to guide and control the quality of the final ontology by heuristically evaluating the choices available to pattern substitution.

3.3.1 Patterns and Ontological Primitives

The first step is to define the target ontological primitives that will be used to transcribe the local patterns into the final ontology. As previously stated this work does not produce axioms, it only produces concepts and relations, as many other works on ontology inference do.

The source language is a regular language. From this it follows that there will be a need to represent repetitions and alternatives. Three types of concepts were chosen: sequences; repetitions; abstractions. The first two will have a meta-concept to inherit from and the last one is defined using normal IS-A relations. Besides IS-A relations, two other types will be used: PartOf (*A PartOf B* is used to mean that B is a part-of A) and Optional-PartOf (*A Optional-PartOf B* is used to mean that B is a part-of A, but may not be present in all instances of A). Fig. 3.11 shows a UML inspired diagram for the ontological primitives (the IS-A relation is represented by the blank triangle arrow). Sequence components have an order among them. This is represented by a ternary relation *Preceeds(A, B, C)*, illustrated in Fig. 3.12, where A is a sequence, *A PartOf B*, *A PartOf C* and B precedes C in the sequence. This relation will not be shown in the diagrams in order to keep them simple.

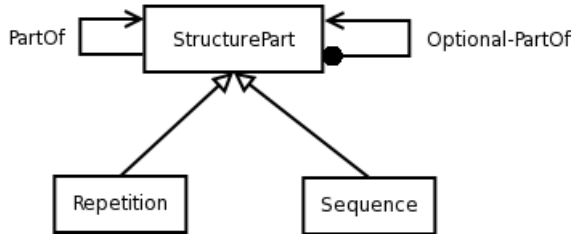

Figure 3.11: Ontological Primitives

Using these ontological primitives, the following describes the initial graph patterns used. Notice that concepts, primitive or otherwise, correspond to alphabet symbols which means that the alphabet size will increase during the transformation.

alternative Given two DFA states A and B (equal or not), if there is more than one transition from A to B, then these are alternative paths between the two states.

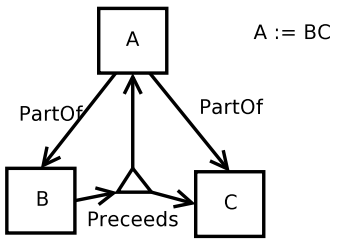

Figure 3.12: Relation Preceeds(A, B, C)

The new concept corresponds to an abstraction gathering the concepts represented by the transition labels. The pattern is illustrated in Fig. 3.13.

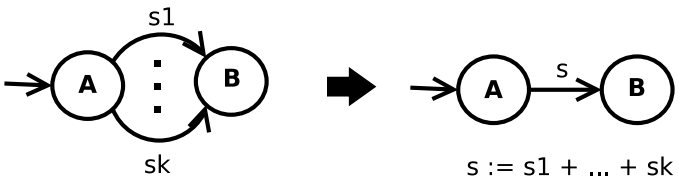

Figure 3.13: Alternative pattern

The new DFA results from replacing the transitions from A to B by a single transition from A to B with the new concept symbol.

optional Consider three states A, B and C, all different, that satisfy:

- there is a transition from A to B and a transition with the same label from both A and B to C,
- B is an non-accepting state if A is a non-accepting state.

Then, the concept that corresponds to the label of the transition from A to B is optional. The pattern is illustrated in Fig. 3.14.

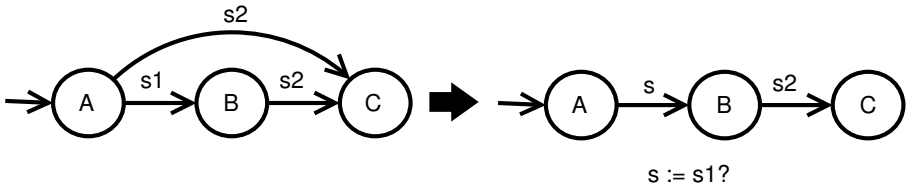

Figure 3.14: Optional pattern

The new DFA results from removing the transition from A to C and then replacing the label of the transition from A to B with a new symbol that marks the concept

as optional. If A and B were both accepting states, then A is no longer an accepting state.

fork Given a state A and a set of states $B_1, ..., B_k$, with $k > 1$. If all states B_i have no outgoing transitions, only have incoming transitions from state A (only one per state B_i) and are either all accepting states or none is, then these are equivalent states. The pattern is illustrated in Fig. 3.15.

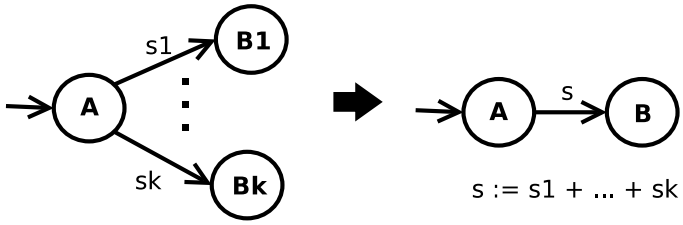

Figure 3.15: Fork pattern

The new DFA results from replacing the states B_i with a single state B and replacing all transitions to the states B_i with a transition to the new state with a new concept symbol that corresponds to an abstraction gathering the concepts represented by the transition labels to the removed states.

sequence Given a set of states $A_1, ..., A_k$ (with $k > 2$) where there is a single transition between each state A_i and A_{i+1}, all states (except, possibly, the first and the last) are non-accepting states, and each state (except, possibly, the first and the last) only has one outgoing transition, then we have a sequence that can be replaced by a single new concept. The pattern is illustrated in Fig. 3.16.

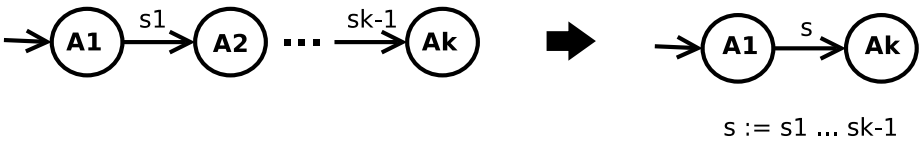

Figure 3.16: Sequence pattern

The new DFA results from removing the states $A_2, ..., A_{k-1}$ and the transition from A_1 to A_2 and adding a new transition labeled by the new concept, from A_1 to A_k.

loop Here two types of loops are distinguished:

simple loop A state A that has a single transition to itself can be replaced by a new concept encoding the repetition of the concept encoded by the label of

the transition. If there are incoming transitions, then we have two situations: either all transitions have the same label and it is equal to the label of the transition from A to A or they don't. If they do, then the repetition can be encoded to say that it is a non null repetition, that is, it must have at least one element. Otherwise it is a normal repetition that can have zero elements. The pattern is illustrated in Fig. 3.17.

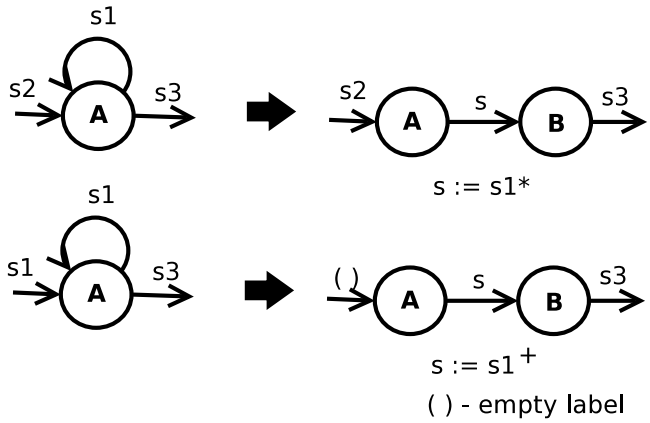

Figure 3.17: Simple loop pattern

The new DFA results from removing the transition from A to A and creating a new state B. State B will be an accepting state iff A was an accepting state. State A will no longer be an accepting state. All other transitions that started from A will now start from B and a transition from A to B will be labeled with the new concept symbol. If it was the case where all incoming transitions to state A had the same label as the transition from A to A, then the labels of those transitions will be replaced by an empty label because those concepts are already captured by this pattern. This change may or may not change the DFA into an NFA.

non-simple loop Consider two different states A and B, such that:

- there is only one transition from A to B and at least one from B back to A,
- A has no other transitions and it is not an accepting state,
- if A is not the initial state, it must have one incoming transition, besides the transitions from B,
- transitions to B, besides the one from A, must come only from states not reachable from B.

The pattern is illustrated in Fig. 3.18.

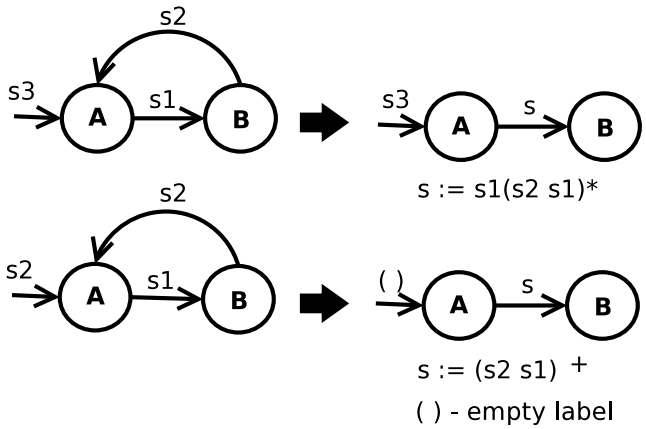

Figure 3.18: Non-simple loop pattern

Similar to the simple loop pattern, if all incoming transitions to A (besides the one from B) have the same label as the one from B, then it is a non null repetition.

The new DFA results from removing the transition from B to A and changing the label of the transition from A to B to the new concept symbol representing the repetition. Incoming links to A (except that from B) are changed in a similar way to the simple loop pattern.

Using these patterns the DFA encoding article structure, built in the previous section, can be transformed into an ontology. Furthermore, these patterns should be sufficient to transform a large set of DFAs. Nevertheless, these patterns were reviewed and changed in order to be as encompassing as possible.

This revision merged the "alternative" pattern with the "fork" pattern and generalized the "optional" and "sequence" patterns. Here are the new versions:

alternative Given state A and a set of equivalent states $B_1, ..., B_k$ (here by equivalent it is meant that their transitions end in the same states and that either they are all accepting or none of them is) such that there are transitions from A to each one of them, then the concepts represented by the labels of those transitions are alternatives. A new concept is created to abstract these alternatives. The pattern is illustrated in Fig. 3.19.

The new DFA results from removing the transitions from A to each B_i, except one. If the initial state of the DFA is one of the states B_i, then that is the state chosen to be kept. A new transition labeled with the new concept symbol is added from A to the chosen state B_i, replacing the existing one. If any of the states B_i no longer has incoming transitions then it is removed from the DFA.

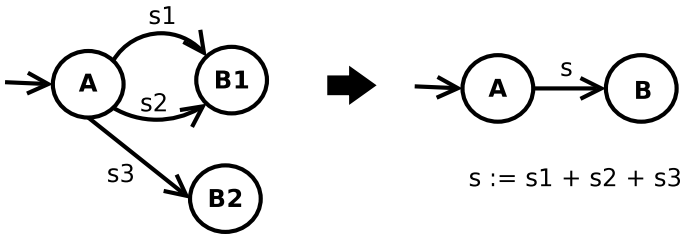

Figure 3.19: Generalized alternative pattern

optional Consider two states A_1 and A_2 such that:

- the only direct connection between them is a transition from A_1 to A_2,
- for each transition starting from A_2 there is one starting from A_1 with the same label and end state,
- A_2 is a non-accepting state if A_1 is a non-accepting state.

Then, the concept represented by the label of the transition from A_1 to A_2 is optional. The pattern in illustrated in Fig. 3.20.

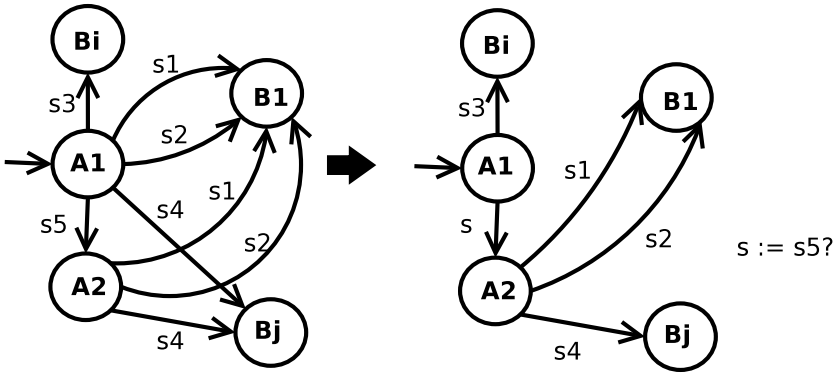

Figure 3.20: Generalized optional pattern

The new DFA results from removing all transitions originating from A_1 that have equivalent transitions (same label and end state) in A_2, except the transition going to state A_2 and modifying the label of that transition so as to mark the concept as optional. If both A_1 and A_2 are accepting states and the only remaining transition in A_1 is the transition to A_2, then A_1 is no longer an accepting state.

sequence The generalized sequence has the same definition and all the conditions, except that inner states can be accepting states. When some of the inner states are accepting states, then we have two cases:

1. If state A_k (the last state in the sequence) has no outgoing transitions, then the sequence can be encoded as a new concept with nested optional sequences. Each optional sequence corresponds to a subsequence starting from an inner accepting state up to the state A_k, with possible inner optional sequences.
2. If state A_k has outgoing transitions, then a new state will be created for each of the accepting states with a transition from A_1 to the new state labeled with the sequence of symbols of the path from A_1 to the state.

3.3.2 Transformation Algorithm

The transformation is clearly a search problem and, as already hinted, there are many equivalent regular expressions for the same language. It follows that there are also many equivalent transformations paths in the sense that they all produce equivalent descriptions using the chosen ontology primitives.

To the end of obtaining an appropriate transformation choice a heuristic cost function was created. The solution cost function attempts to:

- favor small solutions;
- favor a small number of occurrences for each of the primitive concepts (alphabet of the DFA);
- favor concept reuse instead of redefinition;
- favor repetitions over other concept forms;
- favor short sequences.

With this in mind, the function used is the sum of the cost of each concept in the solution. Additionally, concepts are divided into three sets: primitive concepts, regular concepts and anonymous concepts.

Primitive concepts, which correspond to the DFA alphabet, have the following cost function:

$$\text{cost}_{primitive}(x) = \text{count}(x) - 1$$

Here count(x) is the number of times x is used in other patterns of the solution.

Regular concepts correspond to a pattern instance, and their cost is given by:

$$\text{cost}_{regular}(x) = \begin{cases} \frac{1}{K} & \text{if count(x)} = 0 \\ \frac{1}{K \times \text{count}(x)} & \text{if count(x)} > 0 \end{cases}$$

Here K is a factor that favors repetitions ($K = 2$ if the concept is a repetition, otherwise $K = 1$).

Anonymous concepts correspond to auxiliary concepts used to encode a given pattern instantiation. For example, the expressions that correspond to the "non-simple-loop" pattern have internal sequences and these would be described using an anonymous concept. They are defined locally and are not reused because, during search, only pattern occurrences have a name. Their cost, with $K = 2$ as above, is:

$$\text{cost}_{anon}(x) = \begin{cases} 1 & \text{if x is not a repetition} \\ \frac{1}{K} & \text{if x is a repetition} \end{cases}$$

To favor short sequences, there is an additional cost for all sequences with length greater than 2, regular or anonymous:

$$\text{cost}_{seq} = \log(1 + \sum_{x \text{ is seq}} (\text{length}(x) - 2))$$

The total cost function is the sum of all these factors. In Fig. 3.21, we can observe an example ontology and its associated cost. There, repetitions are represented as rounded boxes, to distinguish them from sequences and primitive concepts. The basic concepts are "0", "1" and "2", the regular concepts are "s1" and "s2". "Anon1" is an anonymous concept. The basic concepts "0" and "1" have a cost of 1, because they appear twice in the solution. The concept "s2" contributes with a cost of 1 for being a regular concept that is not used in the definition of other concepts and also contributes with a cost of $\log 3$ because it is a sequence of size 4. The concept "s1" contributes with a cost of 1/2, because it is a repetition and is used in one other concept, namely in "s2".

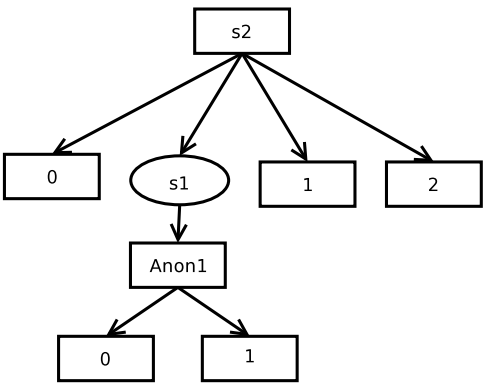

Figure 3.21: Example solution. Cost = 4.977

To actually find the resulting ontology one must use some form of search using the patterns described before as search operators. The objective is to find the sequence of pattern applications such that the resulting DFA consists of only two states with a single transition between them and the cost associated with the sequence, as by the cost function previously defined, is as low as possible.

Given the present search problem, two possible ways present themselves: either use an informed search or use an uninformed search procedure.

Informed searches present two problems here: first the search space for this problem is extremely large and second the cost function is not monotonic. As the search function can decrease as well as increase with search depth, it becomes very difficult to create a good heuristic function to guide the search. That together with the large search space results in an inefficient procedure.

It was decided to use an uninformed search procedure that conserved as much memory as possible, the backtracking depth first search.

Several attempts were made to speed up the search, including limiting the depth by the depth and cost of the current best solution, sorting generated nodes at each step, bookkeeping of previously seen nodes. In the end, the solution chosen combines the bookkeeping of previously seen nodes (for some pattern sequences, permutations give equivalent solutions) using an approximate hash function and the splitting of the source DFA and then combining the solutions found for each piece. The results of the different alternatives explored, using as input the bottom DFA of Fig. 3.22, can be seen in Table 3.15. Data in that table was obtained using an implementation in Clisp 2.33.2 on a 600 MHz Pentium III. All searches produce the same ontology structure, when choosing the best result evaluated by cost heuristic previously described. It can be observed that the most effective reduction is caused by bookkeeping of previously seen nodes. Also, using the new generalized patterns (row F of the Table), the search space is indeed enlarged at the cost of a larger search time. As such, if the problem allows, it is advisable to use the first version of the operators.

Splitting is done recursively. At each step, the DFA is broken into two by choosing a state such that all paths from states on one side of the split to the other side must go through the selected state. The splitting node will be part of both sides and will be made an accepting state or the initial state as appropriate. In the end, if one or more consecutive parts have just two states, then they are merged back together and with one of the remaining parts.

Observe that if the splitting state has incoming transitions with the same symbol from both parts, then in one of the parts (the one where the splitting state is the end state) the transitions will became epsilon transitions[6] and on the other part a new state will be created with a transition to the split state (that is kept by both parts) using the symbol mentioned. This is illustrated in Fig. 3.22.

[6]An epsilon transition is an empty transition, an alphabet symbol is not consumed by its transversal.

Table 3.15: Search function evolution (Backtrack search)

	Solution depth			Branch factor					
Search	Min	Max	Avg	Min	Max	Avg	Expanded	Generated	Time (sec.)
A	12	16	15.65	1	6	1.5	148152	212384	143
B	10	14	10.83	1	6	1.6	780509	1229147	1104
C	10	14	10.82	1	6	1.6	767614	1215108	1452
D	10	14	10.83	1	6	2.4	1298	3148	7
E	12	14	13.39	1	6	2.3	1487	3361	6
F	12	21	17.28	1	6	2.6	27389	70270	175

A – Maximum of 10000 solutions

B – Depth limited by previous best solution

C – Same as B with nodes sorted

D – Same as C with node bookkeeping

E – Only node bookkeeping

F – Node bookkeeping and new generalized patterns

3.3.3 Document Inference Results

The method described was applied to the results of the L* approach, shown in Fig. 3.5. The same alphabet is used (repeated here for convenience): ConferenceTitle (0), Title (1), Author (2), AbstractTitle (3), AbstractText (4), IndexTitle (6), IndexText (7), SectionTitle (8), SubSectionTitle (10), SubSubSectionTitle (11), SimpleText (5), FormatedText (9), FigureCaption (12). The split part of the transformation process produces the intermediate DFAs shown in Fig. 3.22.

Fig. 3.23 shows the ontology generated by this method for the first part of the DFA. Note that order relations are omitted, but display order corresponds to the order in the sequence. Ellipses are used to represent concepts of the "repetition" type while rectangles are used for sequences and primitive concepts. Primitive concepts are labeled with their corresponding alphabet numbers. Other concepts are either named with a leading "s", for concepts that correspond to patterns, or with a leading "Anon", for concepts that were created as auxiliary concepts in the construction of one of the pattern concepts.

Fig. 3.24 shows the ontology generated by the merger of the result of each part of the original DFA.

The resulting ontology contains new concepts that need to be named. Starting from the bottom of Fig. 3.24, *s1* abstracts the various types of text content that appear inside sections (SimpleText (5), FormatedText (9) and FigureCaption (12)) and as such could

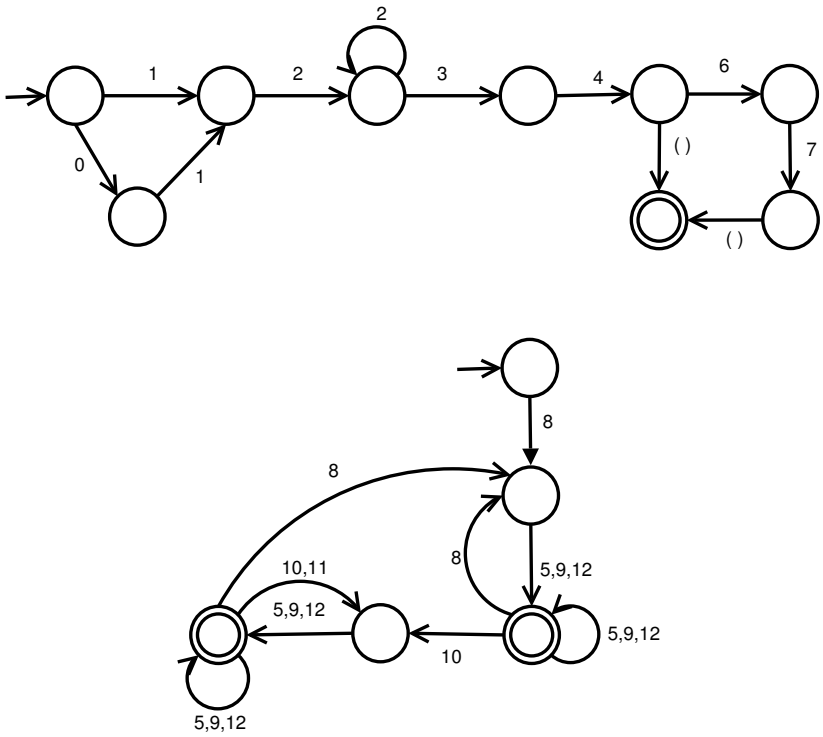

Figure 3.22: Document DFA split into two parts [( ) represents an empty label transition]

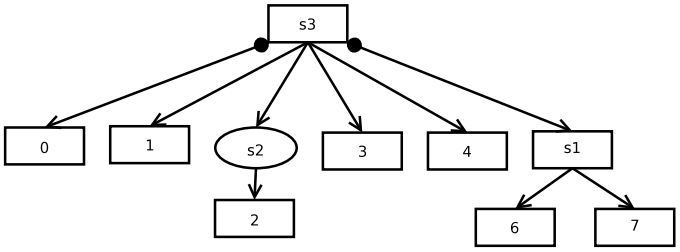

Figure 3.23: Ontology generated from the first part of the DFA

be simply named Text. Next, *s2* is a sequence of Text elements, that is used as the actual content of section, and could be named as TextSequence or SectionText. Each section level, section, subsection and subsubsection share a similar structure, and are represented respectively by *Anon5*, *Anon4* and *Anon2*. The section and subsection have a concept that represents the sequence of their internal elements, *s5* and *s3*, respectively, that could be named SectionContent and SubSectionContent, respectively. All section levels have a corresponding concept that represents their possible repetition, *s6*, *s4* and *Anon1*. Finally, we have *s7* that groups the IndexTitle and IndexText, and could be named Index, *s8* that represents a sequence of Authors and *s9* that groups all the concepts forming an article. The labeled ontology can be seen in Fig. 3.25.

Although this is a reasonable structural description for an article, and one that is in

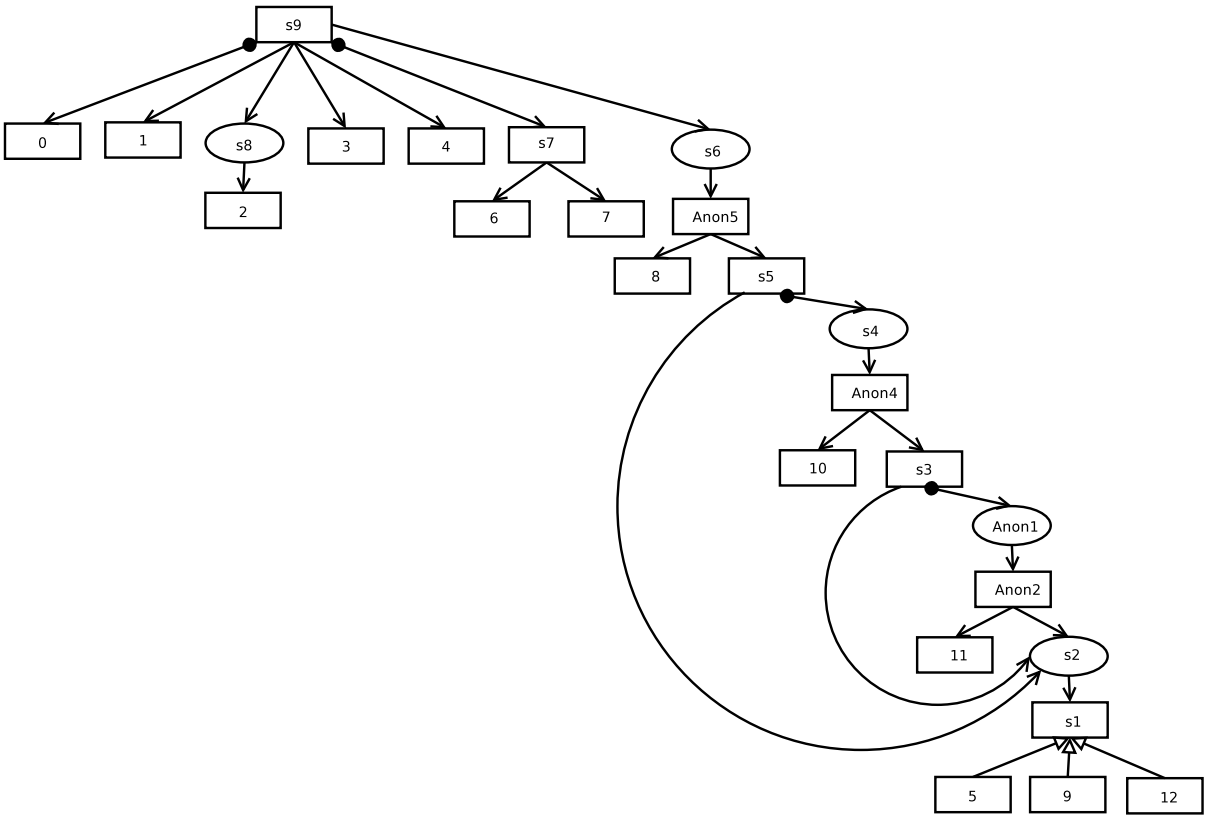

Figure 3.24: Ontology generated for the full DFA

accordance with the principles set in the results part of Section 3.2.1, different structural descriptions are also possible. For example, although the IndexTitle and IndexText are grouped together, the same did not occur with AbstractTitle and AbstractText. Another distinction from the usual handmade ontologies is the relative lack of IS-A relations. For instance, it is possible to imagine grouping the various "titles" under a "Title" concept, or even the creation of a generic "section" structure that would be inherited by the different section levels, as they all share a similar structure.

A handmade ontology might also feature concepts for figures, tables and footnotes. Also, some often used particular cases of sections could be added to the ontology, such as the introduction and references section. These shortcomings are not due to the method, as these extra concepts are not present in the primitive concepts or the DFA formulation.

3.3.4 Comparison with Dublin Core Metadata

The Dublin Core Metadata (DC) standard was created by the Dublin Core Metadata Initiative[7] to be a well defined set of terms to use as metadata description for resources.

Metadata, in the sense employed in the DC, consists of a set of attributes, or elements,

[7]http://dublincore.org

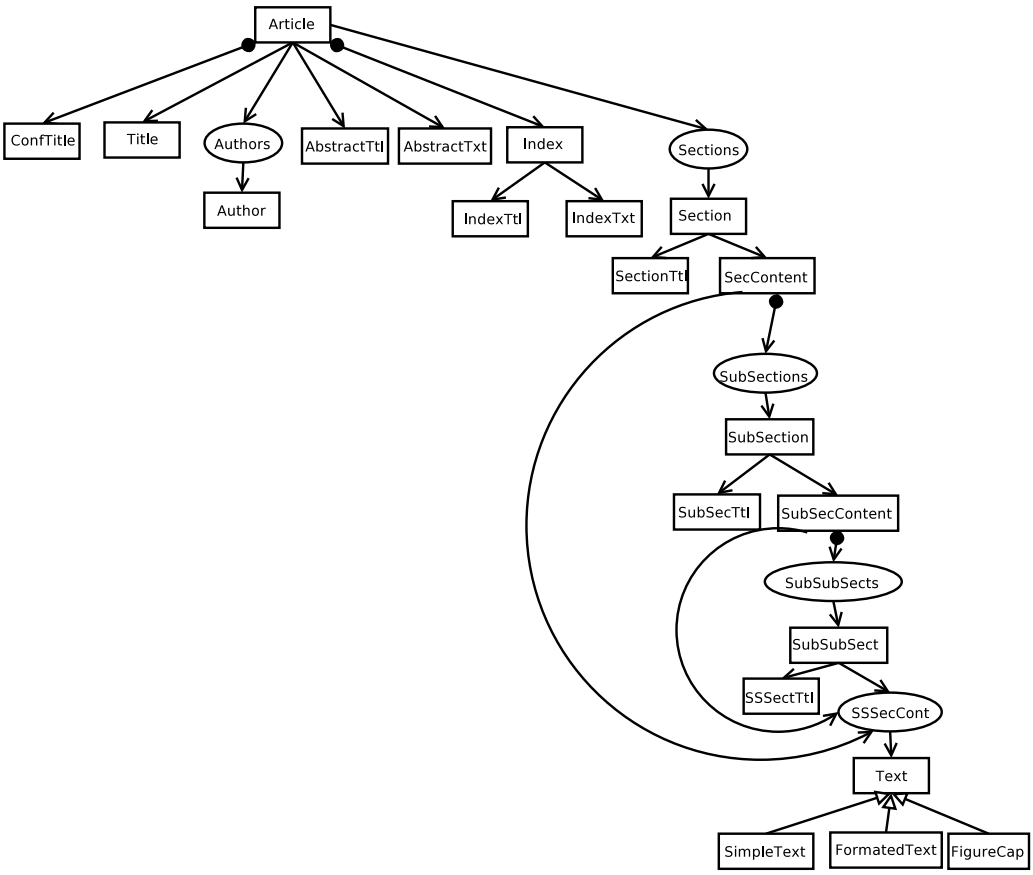

Figure 3.25: Ontology generated for the full DFA with labeled concepts

that describe a resource. For example, a library catalog system would be used to describe the publications in the library according to a pre-defined set of attributes such as: author, title, date of creation or publication, subject coverage, and the call number specifying the location of the item on the shelf.

Metadata can be either embedded in the resource, such as the information contained in the verso of a book's title page (with publisher information, copyrights, ISBN, author, edition, etc) or it can be in a separate record, as in the library catalog. The DC standard does not prescribe either option, leaving the decision to the particular implementation.

The DC[8] consists of a set of elements[9] and qualifiers[10]. The elements are simply the terms used to describe the resource. Each term can is optional and can be repeated at will, however the order in which they appear in the description is not guaranteed to be preserved. Element qualifiers are of two kinds:

- Refinements – these are elements that refine by restriction the meaning of a base element;

[8]http://dublincore.org/documents/dcmi-terms
[9]http://dublincore.org/documents/dces
[10]http://dublincore.org/documents/usageguide/qualifiers.shtml

- Schemas – these are used to aid in the interpretation of the element value, such as controlled vocabularies and formal notations or parsing rules.

Qualifiers are defined so that a client can ignore any qualifier, use the value as if it were unqualified and still be generally correct.

Although the DC is used to describe resources, among them articles, it has a different aim than the ontology resulting from the application of the method described in this thesis. Our resulting ontology describes the internal structure of the type of document to which it was applied, while the DC describes information about a resource, document or not, but treats it as an opaque object and is not concerned with its internal structure.

Albeit their differences, the DC and our resulting ontology share some common elements (concepts). These DC elements that have some similarity with ours are:

- Title – this corresponds to the Title;
- Creator – this corresponds to the Author;
- Description (and its refinement Abstract) – this corresponds to the AbstractText;
- Subject – this would correspond to the keywords in an article, although in our experiment these were not considered;
- Type – this does not have a clear translation, but could be assigned to the root concept, Article;
- tableOfContents (a refinement of Description) – this could be mapped to Index;

There is an "isPartOf" relation defined in the DC, however it does not have the same meaning as our "PartOf" relation. The DC "isPartOf" relation refers to the inclusion of a independent resource within another, such as the inclusion of articles in a article collection, while our "PartOf" relation refers to the internal decomposition of part of an article, something that does not have an existence as an independent unit.

Although the concepts in our ontology that have a mapping in the DC may not provide an adequately complete description of the articles that are parsed according to our ontology, it may nonetheless make sense to use such a mapping as an aid to complete such a description.

3.3.5 Parsing Problem

After ontology extraction one needs to obtain ontology instances from source documents. This is necessary, for example, for page annotation, to enable the use of search methods on large sets of instances.

From the chosen ontology encoding, and because this is all obtained from a regular language, it is trivial to transform into a regular grammar. However, the transformation into grammar production rules, and latter parsing, of repetitions and sequences needs a little care. When transforming a repetition, for example the concept *s2* in Fig. 3.23, a typical transformation would be

```
s2 -> 2 s2
s2 -> 2
```

that would result in many occurrences of *s2* as a non-terminal in a parse tree. During the conversion of the parse tree into concept instances, this must be taken into account and only generate a single concept instance for *s2*. As to the transformation of sequences, if restricted to a regular grammar, then a set of stand-in non-terminals must be used, that will not have a translation back into concept instances. For example, a possible way to encode the concept *s1* would be

```
s1 -> 6 t1
t1 -> 7
```

and where *t1* is one such stand-in non-terminal. Similarly to repetitions, when converting a parse tree into concept instances, this must be taken into account and concept instances under the non-terminal *t1* must be taken as being under the corresponding non-terminal *s1*. The problem faced with sequences is not present if a context-free grammar is used, as these do not impose restrictions on the form of the right-hand side of the grammar production rules.

As an example, the ontology of Fig. 3.23, both in a regular grammar form (in the left column) and a context-free form (in the right column), would be:

```
s3 -> 0 t1                s3 -> 0 1 s2 3 4 s1
s3 -> 1 s2                s3 -> 1 s2 3 4 s1
t1 -> 1 s2                s3 -> 0 1 s2 3 4
s2 -> 2 s2                s3 -> 1 s2 3 4
s2 -> 2 t3                s2 -> 2 s2
t3 -> 3 t4                s2 -> 2
t4 -> 4 s1                s1 -> 6 7
t4 -> 4
s1 -> 6 t5
t5 -> 7
```

In grammar form, it is possible to use a parser to obtain a parsing of the document. Non-terminals in the parse tree corresponding to concepts allow for the creation of in-

stances of those concepts and the appropriate relations from them to lower level concept instances, right down to the primitive concept instances (the string alphabet).

A parse tree for the string "1 2 2 3 4" and the corresponding set of ontology instances, represented by rounded boxes, can be seen in Fig. 3.26.

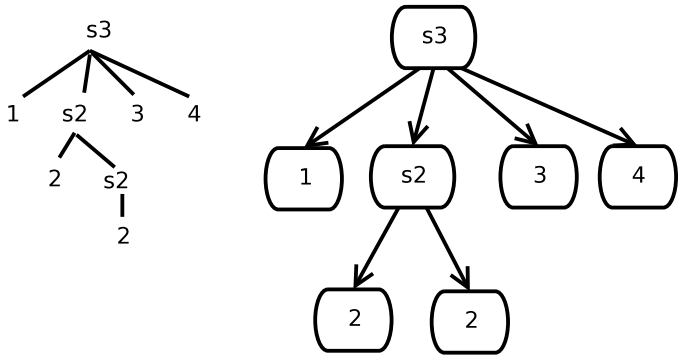

Figure 3.26: Parse tree and ontology instances for string "1 2 2 3 4".

Moreover, with an error-correcting parser, such as [73], some of the classification error that appear from the tokenization step, could potentially be resolved. If the DFA obtained from one of the learning approaches is a correct representation of the document structure, then errors present in documents due to the tokenization step may be detected by an error-correcting parser. It is conceivable that such errors would be correctly resolved with the error-correction mechanism of the chosen parser, although no such study was done in the course of this work.

3.3.6 Discussion

The approach described for the transformation of DFA into the defined ontology representation results in an ontology that correctly represents the principles set when the DFA was created. Moreover, the formulation of the different section levels is correct despite it being a difficult transformation, especially the subsection and subsubsection whose encoding overlaps in the DFA. This works, not only because of the patterns chosen, but also because of the cost heuristic that was presented.

There is a restriction in the foregoing approach, namely that the chosen patterns are not capable of coping with the full class of DFA. Identifying the effective class covered by this approach is an open question. If the sections of a DFA that cannot be transformed by this approach form an isolated subgraph, then it would be conceivable to apply to that subgraph the standard algorithm for DFA to regular expression conversion and somehow merge the results with the remainder of the transformation.

Chapter 4

Conclusions and Future Work

Individual document types, such as technical papers, have a recognizable implicit structure. To enable the use of structure information, it must be made explicit. This work presents a method to infer this structure and encode it into an ontology. A side effect of such a transformation is to allow for automatic retrieval of ontology instances, that is, to allow for the automatic identification of the structure elements of each document and to map them to the concepts in the inferred ontology. Having such an ontology and a set of documents mapped to it should enable more elaborate searches to be performed on them, as well as improve precision and recall values of searches by allowing them to focus on specific parts of the documents.

The approach developed in this work for learning ontologies about the structure of technical papers, differs from existing work in web page wrapper induction by not requiring that source pages have a fixed structure. It also differs from most previous work on ontology learning, by inferring an ontology about the structure of documents from samples of documents.

The method is divided into three parts:

Tokenization The tokenization step, presented in Section 3.1, aimed at the identification of a set of primitive concepts to be used in the second step. It involved the development of a parser to segment the source documents using a set of pre-defined features that were extracted from the documents formatting. A Naive Bayes classifier was employed to identify each of the primitive concepts, linking them to document segments. The parsing approach used for HTML documents, could in fact be used for non-HTML sources, since the feature set is largely HTML independent. Still regarding the tokenization step, a few of the primitive concepts revealed high error rates, despite the overall good results. The work on the tokenization step was published in [2].

Language Learning The language learning step aimed at the inference of an DFA en-

coding the structure of the document. Both an active learning approach and a MDL based approach were presented.

The active learning approach, proposed in Section 3.2.1 and based on the well known L* algorithm, provided an effective reduction in the number of queries required to learn a language, enabling (or at least taking a big step) towards the use of such an algorithm with human interaction. This was accomplished by proposing a new, more powerful teacher, that justify negative answers by selecting the relevant parts of the rejected string. It still leaves a somewhat large number of queries to be answered and the question of how to approach the presentation of conjectures to the user and input of counter-examples. To a person not familiar with the DFA formalism it may not be very useful to present the conjecture in that form. A possible approach could be to present the result of transforming the DFA into an ontology, based on the work described below, assuming that the person is more familiar with ontologies since it is using an ontology learning method. The work regarding the use of a more powerful teacher with the L* algorithm was published in [3].

The remaining work, described below, is still to be published.

The second approach to language learning, presented in Section 3.2.2, is a proposal for a formulation of the MDL principle for DFA inference. Although conditioned by the difficulty presented by a large search space, that limit its scalability to large problems, it is able to infer the expected DFA using only positive examples. In addition, due to an imperfect estimation of model complexity, it may take a larger number of samples than an analytical solution would require to infer a given DFA.

Ontology Building The ontology building step aims at transforming the DFA description of the document structure into an ontology. In Section 3.3, the proposed transformation provides an effective method which results in a clear and concise description. To accomplish this, a recursive pattern substitution algorithm was employed that provided a bottom-up method to construct the ontology from the DFA description of the source document structure. It does not cover the entire regular language class, but should be sufficient for the application in question, perhaps being augmented with a more general algorithm if the need arises.

Overall, the proposed method provides for a straightforward transformation from technical papers to an ontology about their structure. However, some manual work is still needed: the construction of a labeled training set for the Naive Bayes classifier in the first step; answering questions if the active learning approach is used in the second step; labeling inferred concepts at the end of the third step. If it is possible to apply the MDL

approach in the second step, the only remaining tedious manual work is the labeling of the training set for the tokenization step.

Several opportunities for further improvement exist. The tokenization step still presents a high error rate in some classes, leaving room for improvement. In the active learning method, the use of a human teacher would still require some investigation into the problem of presenting and answering equivalence queries. In the MDL approach, there is ample space to improve the search method and perhaps the model complexity estimation, or even to obtain an algebraic expression for the model complexity. Finally, in the ontology building step, the regular language subclass to which this method applies should be determined.

An additional avenue for future work is the integration of the several components of the developed method into a full application providing an efficient storage for documents and a search interface, besides a graphical interface for the ontology learning method per se.

Appendix A

Implemented Software

The ontology learning method described in this thesis was implemented using several programming languages: C#, C and Common Lisp. All the software modules can be used both in Windows and in Linux (using Mono[1], a .Net implementation for Linux.

As mentioned in the section describing the tokenization step of the method, the preprocessing of source HTML pages is done using the HTML Tidy[2] program, that is available for several operating systems.

The learning method has three steps and for each a set of programs were developed.

Tokenization The tokenization step has as source articles in XHTML format. The algorithms for segmentation and classification were implemented in C# in a library called "SegmentosCore.dll". This library provides all the necessary functions and is then used by the following utilities:

segmenter.exe This utility divides the source documents into segments and has four arguments: *source folder*, where the XHTML files are located; *file pattern*, used to identify the files, for example "*.xhtml"; *target folder*, where the resulting XML files are stored; *keyword file*, where the list of keywords to be used in the *Keyword* attribute is stored.

trainer.exe This utility performs the training of the Naive Bayes classifier from a set of XML files with labeled segments. The resulting XML file contains the information required to run the classifier on new documents. It takes two arguments: *source folder*, where the XML files are located; *file pattern*, used to identify the files, for example "*.xml".

classify.exe This utility performs the segment classification, using the XML files from the *segmenter* utility and the information from the *trainer* utility. It takes

[1] http://www.mono-project.com

[2] Available at http://www.w3.org/People/Raggett/tidy/

two arguments: *XML document file*, a XML file resulting from the segmentation of a XHTML document; *classifier data*, a XML file with the information from the training of the Naive Bayes classifier.

stringgen.exe This utility converts a document into a string usable by the MDL learner. Each document to be used in the MDL learner must be converted with this utility and the resulting strings stored in a single text file. The utility takes two arguments: *XML document file*, a XML file with the segments classified; *concept table*, this optional argument is an ordered list of the primitive concepts, where the position of a concept in the list will be its number in the resulting string.

Language learning For the language learning step two different approaches were developed, an active learning approach and an MDL based approach. Both algorithms were implemented in C# libraries and a common library "common.dll" was created that contained a neutral DFA representation to be used as output for both approaches.

The active learning approach was implemented in a C# library called "lstar.dll" that is used by the utility **learner.exe**. This utility does not have any input, as it interacts with the user to learn a DFA. In the end it outputs a file with a representation of the DFA.

The MDL based learner is implemented in C to be as efficient as possible, and is compiled into a library named "libmdl.so" on Linux and "libmdl.dll" on Windows. To make the algorithm available in C#, a wrapper library called "MDL.dll" was created that interfaces with the library written in C. The C# library uses the "common.dll" library to provide an output in a common format.

To invoke the MDL based learner, a simple utility was written in C#, using the wrapper library. This utility, called **mdlRun.exe**, takes two arguments: *string file*, a file with the input strings; *complexity table*, a pre-calculated table with model complexities for the alphabet size of the input strings. In the end it outputs a file with a representation of the DFA.

Ontology building The ontology building step takes an DFA from one of the previous language learning methods and creates a file with the description of the resulting ontology. This step is implemented in Common Lisp, and it is our plan to reimplement it in C# sometime in the future.

The algorithms are implemented in a series of Lisp files. However, there is a function used to initiate the process, **run-file**. This function takes three arguments: *DFA file-*

name, the filename of the DFA to use as input; *search function*, the search function to use, for example `dfa-split-search` that implements the chosen search procedure; *show solutions*, an optional boolean parameter that indicates if the method is to print the solutions when it is finished.

To the exception of the last step, all the algorithms are implemented as C# libraries. This was done to permit the future integration of them into a single application that provides a friendly interface for the learning method.

Appendix B

Published Papers

1. André Luís Martins, H. Sofia Pinto, and Arlindo L. Oliveira. Towards Automatic Learning of a Structure Ontology for Technical Articles. In *Semantic Web Workshop at SIGIR 2004*, 2004.

2. André Luís Martins, H. Sofia Pinto, and Arlindo L. Oliveira. Using a More Powerful Teacher to Reduce the Number of Queries of the L* Algorithm in Practical Applications. In C. Bento, A. Cardoso, and G. Dias, editors, *Proceedings of EPIA 2005, volume 3808 of Lecture Notes in Artificial Intelligence*, pages 325–336. Springer-Verlag, 2005.

Full texts of these papers are given in the following pages.

Towards Automatic Learning of a Structure Ontology For Technical Articles

André L. Martins*
almar@@algos.inesc-id.pt

H. Sofia Pinto*
sofia@algos.inesc-id.pt

Arlindo L. Oliveira*
aml@algos.inesc-id.pt

*INESC-ID/IST
Av. Alves Redol, 9. 1000-029
Lisboa, Portugal

ABSTRACT

Despite the high level of success attained by keyword based information retrieval methods, a significant fraction of information retrieval tasks still needs to take into account the semantics of the data. We propose a method that combines an hand-crafted ontology with a robust inductive inference method to assign semantic labels to pieces of technical articles available on the Web. This approach, together with a query language developed for the purpose, supports queries that cannot be resolved using currently available tools. We present preliminary results that describe the precision of the assigned labels and the accuracy of the replies to the semantic queries present to the system.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence; I.2.6 [**Artificial Intelligence**]: Learning—*Concept learning*

General Terms

Algorithms

Keywords

Ontology, ontology learning, naive bayes, information retrieval

1. INTRODUCTION

There are billions of documents available on the Web about many different subjects, ready to be retrieved by search engines. Search engines use several techniques, but the most important is keyword search. Usually too many documents are retrieved. Therefore, it would be interesting to narrow down the number of documents that are retrieved by using some sort of meaning of the information present in those documents – the semantics. For instance, we may be looking for conferences where Mr. X is engaged as Chair, but not in the Program Committee. If we search, using the keywords "Conference Chair" and "X" we are bound to retrieve, not only the pages of the Conferences where he is Chair, but also Conferences where he is in the Program Committee and even its CV.

In the case of documents, in particular articles, we may want to retrieve documents whose author is Mr. X, but where Mr. X is not referred in the bibliography or vice-versa. In this case, the semantics is connected to the part of the document where the information is present. Therefore, the meaning of the information present in a webpage is important to improve search engines. If we could combine keyword search with some sort of semantic search we would be able to improve precision and recall.

One of the aims of the Semantic Web vision [3] is to use the semantics of the information in the web to allow more sophisticated and precise queries.

Ontologies provide a shared and common understanding of a domain that can be communicated between people, and heterogeneous and widely spread application systems. Typically, ontologies are composed of a set of terms representing concepts (hierarchically organized) and some specification of their meaning. This specification is usually achieved by a set of constraints that restrict the way those terms can be combined. The latter set constrains the semantics of a term, since it restricts the number of possible interpretations of the term. Therefore, we can use an ontology to represent the semantic meaning of a given term.

If the documents are annotated in accordance with a document ontology it is possible to develop search engines that use this annotation to achieve better results. For instance, if we had information about title, authors, sections and other parts of the document, we would be able to develop a specialized search engine where we could specify that we wanted to find articles about an author whose name contains the keyword "Cook" but we are not interested in documents about cooking – in this case with this keyword in the title or in the content of the article.

In this paper we propose a framework that supports this type of queries.

This paper is organized as follows: We briefly describe the problem and related work(section 2), and the approach we used to solve it (section 3). The preliminary results (section 4) are discussed (section 5). We end with conclusions and future work (section 6).

2. PROBLEM STATEMENT AND RELATED WORK

The extraction of semantic information from semi-structured Web pages is a subject that has been extensively studied in recent years. An exhaustive description of the approaches is outside the scope of this paper, and can be found elsewhere [10]. One possible taxonomy of these approaches classifies them into one the following categories: syntax tree based approaches; approaches based on special purpose languages; interactive wrapper induction; natural language processing methods; ontology based approaches and machine learning based methods.

Syntax tree based approaches use HTML parsers to derive a syntax tree and then process this tree using pre-specified rules. A representative example of this approach is RoadRunner [5].

Wrapper development can also be done using special purpose languages that simplify the process of data processing as, for example, [2].

Interactive wrapper induction tools derive rules for extracting relevant information by interacting with the user [12, 1].

Tools that use natural language processing (NLP) to extract relevant pieces of information from texts [4, 13] apply NLP techniques to derive applicable filters.

More directly related with our work are tools that use handcrafted ontologies to identify relevant pieces of information from texts [7, 6] and tools that use machine learning based approaches to learn from labeled data [9, 8, 11].

Our work is different in that it represents a fusion of the ontology-based approaches and the machine learning methods. Our objective is to use the basic concepts of an ontology as a starting point and then to use labeled examples and an inference method sufficiently powerful and flexible to deal with irregular structures, a capacity that previous methods [9, 8, 11] do not exhibit.

Since we are interested in improving the task of finding relevant articles in the Semantic Web, we require an ontology about documents, and, in particular about articles. An ontology about articles defines concepts, such as title, author, abstract, affiliation, section and its possible subdivisions, bibliography, name of the conference, etc. Moreover, we know that there are relations, and specially spatial relations between these concepts. For instance we know that an abstract comes before any section.

Although we could manually build this ontology, ontology building is still more of an art than an engineering task. Moreover ontology building is a knowledge intensive and time consuming task. On top of that, this would probably mean that we had to manually annotate documents according to it, which is a tantalizing task.

Therefore, the idea is to automatize as much as possible the task. The focus of this paper is the automation of the task of building an ontology about articles using a Naive Bayes approach to identify re-occurring features in HTML documents.

These ideas generalize the range of applicability of the method to pages generated by humans using a format that is not necessarily consistent. In this respect, our approach is more powerful than previous ones, that were able to handle only machine generated pages. There are also some common points with other approaches, namely the pre-processing stage, common to most HTML-aware extraction tools.

3. FEATURE EXTRACTION

The focus of the work reported in this paper is the decomposition of the text of an article into individual segments. Each segment is characterized by a set of attribute-value pairs and assigned to a particular class.

This section describes the proposed approach and techniques.

3.1 General Approach

The approach used to perform the segmentation of HTML articles is divided into three stages: (1) a pre-processing stage, followed by the (2) segmentation stage and finally a (3) classification stage. As a result of this process, each HTML article is broken into instances of a small number of chosen classes.

3.1.1 Pre-Processing

In this stage HTML source files are transformed into XHTML files. This transformation is common to other work on HTML files and is mainly due to two reasons: (1) one needs to clean the usual mistakes that can be found on most HTML files; (2) one can leverage on existing tools that process XML files.

3.1.2 Segmentation

The segmentation stage transforms XHTML files into a list of segments. A segment is a block of text with a set of characterizing attribute values. Each attribute represents an important feature useful either for the segmentation stage or the classification stage.

The set of attributes is divided into three types:

1. XHTML based types - are computed from the inspection of XHTML tags.
   (a) Segment divider - are used to drive segmentation stage. Their change indicates the end of a segment and the beginning of a new one.
   (b) Non-segment divider
2. Post-Processing - are computed from the segments resulting from the segmentation stage.

In general, the attributes give information about three features of segments:

Text Format These are the attributes that drove the segmentation. In our opinion, these attributes are one of the main means by which people identify the different parts of an article structure.

- **Font size (segment divider, post-processing)** The current font size. The font size attribute is normalized by dividing it's value by the maximum found in the current article and then converting the result into an integer from 1 to 10. This allows for a better comparison between articles that use a different scale of font sizes on section titles, for example.
- **Font style (segment divider)** The current font style, that is, if the text is in bold, italic, etc.

Text Information A set of attributes that reflect information contained in the segments text. All of them are post-processing attributes.

- **Text information** The length of the text in each segment has a big weight on its classification. This information is transformed into the following discrete scale: 0 to 30, 30 to 80, 80 to 160, 160 to 1600, 1600 to 16000, bigger than 16000. This scale was obtained empirically. Initially, the first three intervals were collapsed and where intended to capture the size of a one or two line sentence, for example the title of an article. Later we observed that it was helpful to further subdivide this interval, so that small segments of text where not mistaken as titles and section titles. The other intervals are there to capture small paragraphs and larger segments of text.
- **Keyword** If the text segment starts with one of a given list of keywords, the attribute value is the keyword index in the list.
- **Numbering** Some segments may start with a sequence of numbers, namely the section and subsection titles. This attribute captures the cardinality of that sequence. If a section starts with "1.4.5" the attribute value is 3.

Context Another important factor in the classification of a segment is context information which reflects the segments surrounding the segment we are considering. This allows the segment's neighborhood to influence its classification. Here we chose to use:

- **Surrounding text (non-segment divider)** This attribute indicates the presence of surrounding text. This is useful to distinguish between bold-faced font that is used to give emphasis to a block of text and a bold-faced font that is used in a title of a section, for example.
- **Image exists (non-segment divider)** This attribute indicates the existence of an image. It's used to distinguish a image caption from other headings.
- **Next segment Image exists (post-processing)** The value of the "Image exists" attribute of the following segment. This was found to be useful in recognizing figure captions that sometimes occur before the image.
- **Class of previous segment (post-processing)** This attribute contains the classification of the previous segment (this attribute is added during the training and classification stages).
- **Table exists (post-processing)** The same as the "Image exists" attribute but for tables.

3.1.3 Classification

To classify the segments found after the segmentation stage we used a Naive Bayes classifier. In this stage each segment is labeled with the result of the classification. The classification uses the information contained in the attribute of the segment.

The choice of a supervised classification method, versus an unsupervised one, was mainly due to the goal of later creating an ontology. If we had chosen an unsupervised method we would still have to give appropriate class names to the resulting classes.

3.1.4 Naive Bayes Classifier

In a Bayes classifier, a problem instance is described by a set of attribute values and its target value (class) is one of a finite set of values. The target value of a new instance is the class with the greatest a posteriori probability given the training examples. That is:

$$C_{MAP} = \underset{C_i \in C}{argmax}\, P(C_i | A_1 = v_{k1} ... A_n = v_{kn}) \quad (1)$$

Using the Bayes theorem:

$$C_{MAP} = \underset{C_i \in C}{argmax}\, \frac{P(A_1 = v_{k1} ... A_n = v_{kn} | C_i).P(C_i)}{P(A_1 = v_{k1} ... A_n = v_{kn})} \quad (2)$$

which is equivalent to:

$$C_{MAP} = \underset{C_i \in C}{argmax}\, P(A_1 = v_{k1} ... A_n = v_{kn} | C_i).P(C_i) \quad (3)$$

The naive Bayes simplifying hypothesis is that the attribute values are conditionally independent given the target value. The resulting classifier is:

$$C_{NB} = \underset{C_i \in C}{argmax}\, P(C_i) \prod_j P(A_j = v_{kj} | C_i) \quad (4)$$

3.2 Algorithms

3.2.1 Pre-processing

To perform the pre-processing of the HTML pages, we used the HTML Tidy program[1]. This program uses a set of

[1]Available at http://www.w3.org/People/Raggett/tidy/

heuristics to try to correct the mistakes found on HTML pages and allows the transformation of HTML pages into XHTML. The program is not always able to transform the input documents, but works for the vast majority.

3.2.2 Segmentation

The segmentation algorithm has two steps. In the first step, the XHTML page is processed resulting in a list of segments. In the second step, this list is refined.

The first step consists in walking the XML DOM (Document Object Model) [14] tree that corresponds to the XHTML page, starting at the $< body >$ node. There are two cases:

1. If the current node is a text node, then it's value is appended to the current segment's text.
2. If the current node is a tag node, then it is presented to the appropriate attributes for processing. If there is a change in the attributes that drive the segmentation, then a new segment is created with the accumulated text and the current values for the attributes. The tag that caused the segmentation is memorized along with the value for the attributes prior to the change.

When going up the tree, if we pass a tag that was memorized in the second case, the attribute values are restored, as the tag is no longer affecting the remaining text.

The second step is responsible for four things:

1. Merge consecutive segments that have the same attribute values.
2. Remove segments that have only whitespace as text. This sometimes happens because of the use of whitespace to promote page alignment.
3. Apply the attributes that are derived from the segments (post-processing type).
4. Normalize attributes that require it. In this work the only attribute normalized is the font size, as previously explained.

The order of the first and second items in this step are important since the "empty" segments are usually text dividers and as such the segments they split should not be merged.

3.2.3 Classification

As it was already said, we used the Naive Bayes classification algorithm. To use this classifier we have to estimate the following probabilities during the training phase:

- $P(C_i)$, where C_i is one of the chosen classes.

$$P(C_i) = \frac{segments_i}{|examples|} \tag{5}$$

  where $segments_i$ is the number of segments classified in C_i and $|examples|$ is the number of segments presented in the training set.

- $P(A_j = v_k|C_i)$, where A_j is an attribute and v_k is one of its values. Here we use the *m-estimate* of probability:

$$P(A|C) = \frac{n_c + mp}{n + m} \tag{6}$$

  where n_c is the number of times A is classified in C, n is the total number of examples of A, p is the prior estimate of the probability we wish to determine and m is a constant called *equivalent sample size*, which determines how heavily to weight p relative to the observed data.

  We used $p = \frac{1}{k}$ as the prior, where k is the total number of different attribute-value pairs that where observed ($|Vocabulary|$) and $m = k$.

$$P(A_j = v_k|C_i) = \frac{n_l + 1}{n + |Vocabulary|} \tag{7}$$

  Here n_l is the number of times $A_j = v_k$ is classified as C_i, n is the number of occurrences of $A_j = v_k$.

The classification phase is done as usual, but ignoring attribute-value pairs that haven't been seen during the training phase.

4. PRELIMINARY RESULTS

4.1 Classifying text segments

The process was tested on a set of 25 HTML articles from several World Wide Web Conferences.[2] The articles were segmented and them manually classified into the chosen classes (Author(s), Title, AbstractTitle, AbstractText, IndexTitle, IndexText, Figure(or Table)Caption, ConferenceTitle, SectionTitle, SubSectionTitle, SubSubSectionTitle, SimpleText, FormatedText). These classes represent most of the structure elements present in articles.

The keywords used were: "abstract", "figure", "table", "contents" and "overview".

Note that the attribute that indicates the classification of the previous segment is updated dynamically during the classification to reflect the result of the actual classification of the previous segment (having a value of "null" for the first segment of each document). During the training phase this attribute is extracted from the previous segment (having a value of "null" for the first segment of each document).

The segmentation resulted in 5472 segments which were then used to perform a ten fold cross-validation. The results are presented in tables 1 and 2.

4.2 Simple search test

To further validate the results of this method, we developed a simple query language (see 5.2.1) that retrieves documents based on segment contents and their classification. We applied this language to a set of 51 documents, retrieved from the WWWC 2003 conference (see 5.2.2). This set of documents was segmented and classified using the previously described classifier trained in a separate set of 25 documents.

[2]WWWC96, WWWC2000, WWWC2001, WWWC2002 and WWWC2003

Table 1: Classification Error

Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Fold 8	Fold 9	Fold 10	Average
7.68%	6.22%	2.38%	3.47%	1.83%	2.01%	0.73%	2.74%	0%	4.02%	3.11%

Table 2: Classification data – target vs. obtained

	CnfT	Ttl	Aths	ATtl	ATxt	FTxt	STxt	STtl	SSTtl	IdxTtl	Idx	SSSTtl	FigC	Error
CnfT	9	0	0	0	0	0	0	0	0	0	0	0	0	0%
Ttl	0	25	1	0	0	0	0	0	0	0	0	0	0	3.85%
Aths	0	0	18	0	0	5	5	4	0	0	0	0	0	43.75%
ATtl	0	0	0	22	0	1	0	0	0	0	0	0	0	4.35%
ATxt	0	0	0	0	13	0	7	0	0	0	0	0	0	35%
FTxt	0	0	9	2	3	2204	17	3	32	0	0	10	1	3.38%
STxt	0	0	0	0	5	17	2511	0	1	0	0	2	0	0.99%
STtl	0	0	0	0	0	0	0	196	3	0	0	0	0	1.51%
SSTtl	0	0	0	0	0	4	0	9	179	0	0	0	0	6.77%
IdxTtl	0	0	0	2	0	0	0	1	0	0	0	0	0	100%
Idx	0	0	0	0	2	0	1	0	0	0	0	0	0	100%
SSSTtl	0	0	0	0	0	2	0	0	12	0	0	35	1	30%
FigC	0	2	0	0	0	3	1	0	2	0	0	0	88	8.33%

ConferenceTitle – CnfT, Title – Ttl, Author(s) – Aths, AbstractTitle – ATtl, AbstractText – ATxt, FormatedText – FTxt, SimpleText – STxt, SectionTitle – STtl, SubSectionTitle – SSTtl, IndexTitle – IdxTtl, Index – Idx, SubSubSectionTitle – SSSTtl, FigureCaption – FigC

4.2.1 Language description

To construct the queries with the segmented and classified text we created a small language to express restrictions on the search using the classification results. The simple query language used is the following.

```
expr: (AND expr1 ... exprN)
    | (OR expr1 ... exprN)
    | (NOT expr)
    | (IN text segs)

segs: seg-expr
    | (FROM seg-expr [seg-expr])

seg-expr: class
        | (WITH text class)
```

The "IN" operator has a value of "true" when some of the segments given by "segs" has the specified text. The "FROM" operator selects the segments that lie between a segment in the first "seg–expr" and the first segment that follows it contained in the next "seg–expr" or the end of the document. Finally the "WITH" operator selects the segments of a particular class that contain the specified text.

4.2.2 Queries

In this section we show a few examples that show how the semantic knowledge acquired can be used to elaborate complex queries not easily expressed in current keyword based technology and show the applicability of our system.

In some cases we did not get the expected results and had to refine the queries. In these cases we present how the initial query was refined until we got the expected results.

The results presented in table 3 show the precision and recall for the following queries:

1. Articles with "Semantic Web" in the abstract text

   ```
   (IN ``Semantic Web'' AbstractText)
   ```

2. Articles with Maedche as an author

   ```
   (IN ``Maedche'' Author)
   ```

3. Articles citing Maedche

   In some articles the reference section is named "Bibliography" while in others "References". Therefore the query was refined to include both cases.

   (a)
   ```
   (IN ``Maedche''
       (FROM (WITH ``References''
                   SectionTitle)))
   ```

   (b)
   ```
   (IN ``Maedche''
       (FROM (WITH ``Bibliography''
                   SectionTitle)))
   ```

   (c)
   ```
   (OR (IN ``Maedche''
           (FROM (WITH ``References''
                       SectionTitle)))
       (IN ``Maedche''
           (FROM (WITH ``Bibliography''
                       SectionTitle))))
   ```

4. Articles citing Maedche and where Maedche is not an author

   ```
   (AND (OR (IN ``Maedche''
                (FROM (WITH ``References''
                            SectionTitle)))
            (IN ``Maedche''
   ```

```
            (FROM (WITH ‘‘Bibliography’’
                        SectionTitle))))
    (NOT (IN ‘‘Maedche’’ Author)))
```

5. Articles that reference documents from 2003

   Some documents have a reference to the year of the conference where the article was published before the introduction and others have it at the end of the document, following a section about the authors. Therefore, the query was refined to search from the beginning of the references until the beginning of the following section.

   (a)
```
(OR (IN ‘‘2003’’
        (FROM (WITH ‘‘References’’
                    SectionTitle)))
    (IN ‘‘2003’’
        (FROM (WITH ‘‘Bibliography’’
                    SectionTitle))))
```

   (b)
```
(OR (IN ‘‘2003’’
        (FROM (WITH ‘‘References’’
                    SectionTitle)
              SectionTitle))
    (IN ‘‘2003’’
        (FROM (WITH ‘‘Bibliography’’
                    SectionTitle)
              SectionTitle)))
```

6. Articles with Guha as an author

   Author is among the classes with the highest error rate. Therefore, in this case, we had to find an alternative formulation to the query using surrounding well recognized classes.

   (a)
```
(IN ‘‘Guha’’ Author)
```

   (b)
```
(IN ‘‘Guha’’
    (FROM Title (WITH ‘‘Introduction’’
                      SectionTitle)))
```

7. Articles citing Guha and where Guha is not an author

```
(AND (OR (IN ‘‘Guha’’
             (FROM (WITH ‘‘References’’
                         SectionTitle)))
         (IN ‘‘Guha’’
             (FROM (WITH ‘‘Bibliography’’
                         SectionTitle))))
     (NOT (IN ‘‘Guha’’
              (FROM Title
                    (WITH ‘‘Introduction’’
                          SectionTitle)))))
```

5. DISCUSSION

Overall the classification error in the preliminary results is good. Nevertheless, there are a few classes that have a very high classification error. We will now attempt to detail the reasons for these error rates.

The IndexTitle and IndexText classes have two problems. The IndexTitle is, in terms of text format, identical to a title, for example a section title. Moreover it's position in the beginning of a document also causes it to be mistaken for an abstract title. The IndexText appears similar to a regular text segment and, when the IndexTitle is mistaken for an abstract title, it is confused with the abstract text, due to the use of the attribute indicating the previous classification.

The AbstractText class, like the IndexText class, is very similar to a regular text segment. Moreover it is not always presented in a single segment, but may appear as several SimpleText and FormatedText segments.

The Author(s) class is difficult to classify because it has many different presentation styles. To correctly recover this information would require, in our opinion, a more detailed processing of the text, that includes the position in the page.

Finally, the SubSubSectionTitle, SubSectionTitle and SectionTitle classes are sometimes confused because of the varying font sizes that are used to present them. This is somehow compensated by the normalization of the font size attribute, but not completely. Additionally in some documents all section, subsection, etc. titles have the same font size.

In spite of a high error rate for classes like Author(s), one can still use the classification to achieve good search results, even if one has to use an alternative formulation for the query. The results corresponding to final formulations of the queries are encouraging.

The problems not solved by query refinement are due to the fact that it is not so easy to identify the end of a section since its type can vary, it may be a section, a subsection, etc. The further development of the ontology may provide the means to express more precise semantic queries resulting in better search results.

Table 3: Query results

Query	Rel	Ret	RelRet	Precision	Recall
1	11	11	11	**100.0%**	**100.0%**
2	1	1	1	**100.0%**	**100.0%**
3.a	4	2	2	100.0%	50.0%
3.b	4	2	2	100.0%	50.0%
3.c	4	4	4	**100.0%**	**100.0%**
4	3	3	3	**100.0%**	**100.0%**
5.a	19	27	19	70.4%	100.0%
5.b	19	23	19	**82.6%**	**100.0%**
6.a	1	0	0	-	0.0%
6.b	1	1	1	**100.0%**	**100.0%**
7	7	7	7	**100.0%**	**100.0%**

Relevant – Rel, Retrieved - Ret
Relevant retrieved – RelRet

6. CONCLUSIONS AND FUTURE WORK

In this paper we show how one can perform a robust decomposition of the text of an article into individual segments that correspond to basic natural parts, such as title, section title, text, etc.

We aim at automatizing as much as possible the task of building an ontology of documents. We use a Naive Bayes approach to identify re-occurring features in HTML documents. Once the text of documents is classified according

to this ontology we can use this classification to perform sophisticated semantic queries beyond the power of simple keyword based search engines. In this paper we present a simple query language and the preliminary results that show the potential of the approach.

As next steps we want to further elaborate the ontology learning process so that we can identify complex concepts, such as a reference, and the relationships between concepts, such as composition relations, and *isa* relations. An additional direction for future research is the application of the methods described in this paper to documents in other formats, namely PDF and Postscript. In this case, the lack of easily usable tags could be compensated by the use of text positioning information.

7. REFERENCES

[1] B. Adelberg. NoDoSE - a tool for semi-automatically extracting semi-structured data from text documents. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA*, pages 283–294, June 1998.

[2] G. O. Arocena and A. O. Mendelzon. Weboql: Restructuring documents, databases, and webs. In *Proceedings of the Fourteenth International Conference on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*, pages 24–33. IEEE Computer Society, 1998.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientif American*, May 2001.

[4] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on on Innovative Applications of Artificial Intelligence, Orlando, Florida, USA*, pages 328–334, July 1999.

[5] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, Roma, Italy*, pages 109–118. Morgan Kaufmann, Sept. 2001.

[6] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, Y.-K. Ng, D. Quass, and R. D. Smith. Conceptual-model-based data extraction from multiple-record web pages. *Data and Knowledge Engineering*, 31(3):227–251, 1999.

[7] D. W. Embley, Y. S. Jiang, and Y.-K. Ng. Record-boundary discovery in web documents. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, USA*, pages 467–478, June 1999.

[8] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(8):521–538, 1998.

[9] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1–2):15–68, Apr. 2000.

[10] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Rec.*, 31(2):84–93, 2002.

[11] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[12] B. A. Ribeiro-Neto, A. H. F. Laender, and A. S. da Silva. Extracting semi-structured data through examples. In *Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management, Kansas City, Missouri, USA*, pages 94–101. ACM, Nov. 1999.

[13] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1–3):233–272, 1999.

[14] W3C. The Document Object Model. http://www.w3.org/DOM.

Using a More Powerful Teacher to Reduce the Number of Queries of the L* Algorithm in Practical Applications

André L. Martins[1], H. Sofia Pinto[2], and Arlindo L. Oliveira[3]

INESC-ID/IST - Av. Alves Redol, 9. 1000-029 Lisboa, Portugal
almar@algos.inesc-id.pt[1], sofia@algos.inesc-id.pt[2], aml@inesc-id.pt[3]

Abstract. In this work we propose to use a more powerful teacher to effectively apply query learning algorithms to identify regular languages in practical, real-world problems. More specifically, we define a more powerful set of replies to the membership queries posed by the L* algorithm that reduces the number of such queries by several orders of magnitude in a practical application. The basic idea is to avoid the needless repetition of membership queries in cases where the reply will be negative as long as a particular condition is met by the string in the membership query. We present an example of the application of this method to a real problem, that of inferring a grammar for the structure of technical articles.

1 Introduction and Motivation

Learning using feedback from the teacher, also known as active learning, is an important area of research, with many practical applications. One of the best known approaches to apply active learning to the inference of sequential models is the L* algorithm. In this work we describe an improvement to the L* algorithm, that strongly reduces the number of queries, in a practical application in the area of ontology learning.

Ontologies provide a shared and common understanding of a domain that can be communicated between people, as well as between heterogeneous and widely spread application systems. Typically, ontologies are composed of a set of terms representing concepts (hierarchically organized) and some specification of their meaning. This specification is usually achieved by a set of constraints that restrict the way those terms can be combined. The latter set constrains the semantics of a term, since it restricts the number of possible interpretations of the term. Therefore, we can use an ontology to represent the semantic meaning of given terms.

In our case, we are interested in learning an ontology about articles. Using the semantic information, provided by the ontology, search engines can focus on the relevant parts of documents, therefore improving precision and recall.

Our starting point is a small set of 13 basic concepts: title, author(s), abstract title, abstract text, section title, simple text, formated text, subsection title,

Figure caption, etc. As an intermediate step towards learning the ontology, we aim at inferring a description of an automaton that encodes the structure of articles. For that we have used query learning. However, existing algorithms for query learning use an exceedingly large number of queries for problems of reasonable size, a feature that makes them unusable in real world settings. In this work we propose a solution to the large number of queries required by the L* algorithm in this and other practical settings.

The remainder of this paper is organized as follows: first, we briefly describe the problem and related work (Sect. 2). We describe, in Sect. 3, the algorithm for query learning that is our starting point, L* . We then describe the approach we used to solve the main problem found when using the L* algorithm, the large number of membership queries that needs to be answered (Sect. 4). Our results are presented and discussed in Sect. 5. We end with conclusions (Sect. 6).

2 Related Work

This work addresses the problem of inferring a regular language using queries and counter-examples. The problem of regular language learning has been extensively studied, both from a practical and theoretical point of view.

Selecting the minimum deterministic finite automaton (DFA) consistent with a set of pre-defined, labeled, strings is known to be NP-complete [1]. Furthermore, even the problem of finding a DFA with a number of states only polynomially larger than the number of states of the minimum solution is also NP-complete [2].

Fortunately, the problem becomes easier if the algorithm is allowed to make queries or to experiment with the unknown automaton. Angluin proposed the L* algorithm [3], a method based on the approach described by Gold [4], that solves the problem in polynomial time by allowing the algorithm to ask membership queries. Schapire [5] proposes an interesting alternative approach that does not require the availability of a reset signal to take the automaton to a known state.

Our work aims at making the L* algorithm more applicable to real world problems, by using a more powerful teacher.

We have chosen a particular problem to apply the proposed approach, that of inferring a regular language that models the structure of a document (in our case, of a technical article). Such a model can later be used to label the different parts of a document, a first step towards the desired goal of semantically labeling the document.

A number of approaches have been proposed to date to the problem of inferring the structure of a document using language models. Additionally, a number of methods have been proposed to efficiently detect structures in sequences of symbols using languages (regular or context-free) as models [6,7,8]. However, these approaches are generally concerned with the identification of repeating structures and not, necessarily, semantic units.

We believe that the application of a grammatical inference method to the inference of semantic units in documents will only lead to interesting results

if a human teacher is involved in the learning process. Automatic learning of semantic structures from (possibly labeled) sets of documents is a worthwhile goal, but is likely to require very large corpora and efficient algorithms that do not yet exist.

Other approaches that do not use languages as models for the text also exist, but are more limited in their potential scope, since they look for local features of the text, such as fonts and formats [9,10,11,12,13]. The techniques used in these approaches can also be useful, but have not yet been applied to our problem.

3 Query Learning of Regular Languages

A regular language can be defined by the set of strings that are accepted by a deterministic finite automaton (DFA), defined as follows.

Definition 1. *A DFA defined over the alphabet Σ is a tuple $D = (Q, \Sigma, q_0, F, \delta)$, where:*

Q is a finite set of states;
$q_0 \in Q$ is the initial state;
$F \subset Q$ is the set of final or accepting states;
$\delta : Q \times \Sigma \to Q$ is the transition function.

A string is accepted by a DFA if there exists a sequence of transitions, matching the symbols in the string, starting from the initial state and ending in an accepting state.

The task of learning a regular language can be seen as learning a DFA that accepts the strings belonging to the language.

Query learning is concerned with the problem of learning an unknown concept using the answers provided by a teacher or oracle. In this context, the concept will be a DFA. There are several types of queries, but here we will focus on those sufficient to learn efficiently regular languages [14], namely:

membership queries the learner presents an instance for classification as to whether or not it belongs to the unknown language;
equivalence queries the learner presents a possible concept and the teacher either acknowledges that it is equivalent to the unknown language or returns an instance that distinguishes both.

Angluin presented an efficient algorithm [3] to identify regular languages from queries, the L* algorithm. This algorithm derives the minimum canonical DFA that is consistent with the answered queries.

3.1 The L* Algorithm

In this section we briefly describe the L* algorithm, in order to be able to present the proposed changes.

The L* algorithm defines a learner, in the query learning setting, for regular languages. This learner infers DFAs from the answers to membership and equivalence queries posed to a teacher. A teacher that can answer these types of queries is referred to as a minimally adequate teacher.

The instances used in membership queries are strings defined over the alphabet Σ. The concepts in equivalence queries are DFAs defined over that alphabet.

The information obtained from membership queries is used to define a function[1] $T : ((S \cup S \cdot \Sigma) \cdot E) \rightarrow \{0, 1\}$, where S is a nonempty finite prefix-closed set[2] of strings and E is a nonempty finite suffix-closed set of strings. The set $((S \cup S \cdot \Sigma) \cdot E)$ is the set of strings for which membership queries have been asked.

The function T has a value of 1 when the answer is positive, that is, when the string belongs to the language of the target DFA, and a value of 0 otherwise. It can be viewed as an observation table, where the rows are labeled by the elements of $(S \cup S \cdot \Sigma)$ and columns are labeled by the elements of E. For example, in Table 1, $S = \{\lambda\}$, $E = \{\lambda\}$ and $S \cdot \Sigma = \{0, 1, 2\}$.

Table 1. Initial L* table at the first conjecture

		λ	} E
S {	λ	0	
$S \cdot \Sigma$ [	(2)	0	
	(1)	0	
	(0)	0	

To represent a valid complete DFA, the observation table must meet two properties: closure and consistency. The observation table is *closed* iff, for each t in $S \cdot \Sigma$ there exists an $s \in S$ such that $row(t) = row(s)$. The observation table is *consistent* iff for every s_1 and s_2, elements of S, such that $row(s_1) = row(s_2)$ and for all $a \in \Sigma$, it holds that $row(s_1 \cdot a) = row(s_2 \cdot a)$.

The DFA $D = (Q, q_0, F, \delta)$ that corresponds to a *closed* and *consistent* observation table is defined by:

$Q = \{row(s) : s \in S\}$

$q_0 = row(\lambda)$

$F = \{row(s) : s \in S \wedge T(s) = 1\}$

$\delta(row(s), a) = row(s \cdot a)$

For example, Fig. 1 shows the DFA corresponding to the observation table shown in Table 2. Note that when a string belongs to both S and $S \cdot \Sigma$ then it is only represented once in the observation table (at the top part). This can be seen in Table 2. To obtain the transition function value for the initial state and

[1] Set concatenation $A \cdot B = \{ab | a \in A, b \in B\}$.

[2] A prefix-closed (suffix-closed) set is a set such that the prefix (suffix) of every set element is also a member of the set.

symbol $0 \in \Sigma$, i.e. $\delta(row(\lambda), 0)$, one must lookup $row(\lambda \cdot 0)$, namely, the line labeled by (0). This line is shown in the top part of Table 2 because $(0) \in S$.

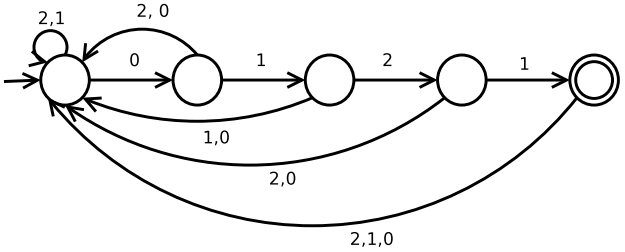

Fig. 1. Intermediate DFA

Table 2. L* table at the second conjecture

	λ	(1)	(2 1)	(1 2 1)
(0 1 2 1)	1	0	0	0
(0 1 2)	0	1	0	0
(0 1)	0	0	1	0
(0)	0	0	0	1
λ	0	0	0	0
(0 1 2 1 2)	0	0	0	0
(0 1 2 1 1)	0	0	0	0
(0 1 2 2)	0	0	0	0
(0 1 2 1 0)	0	0	0	0
(0 1 2 0)	0	0	0	0
(0 1 1)	0	0	0	0
(0 2)	0	0	0	0
(0 1 0)	0	0	0	0
(2)	0	0	0	0
(1)	0	0	0	0
(0 0)	0	0	0	0
(0 2 2 2)	0	0	0	0

It can be proved that any DFA consistent with the observation table, but different by more than an isomorphism, must have more states.

Starting with the initialization of the observation table, the L* algorithm proceeds to build a DFA. The DFA is presented to the teacher as an equivalence query.

Before a DFA can be generated from the observation table, it must verify two properties: the table must be *consistent* and *closed*. Therefore, a loop must be executed until the properties are met, updating the observation table along the way.

If the observation table is not *consistent*, then there exist $s_1, s_2 \in S$, $a \in \Sigma$ and $e \in E$ such that $row(s_1) = row(s_2)$ and $T(s_1 \cdot a \cdot e) \neq T(s_2 \cdot a \cdot e)$. The set E is augmented with $a \cdot e$ and the observation table is extended using membership queries.

If the observation table is not *closed*, then there exist an $s_1 \in S$ and an $a \in \Sigma$ such that $row(s_1 \cdot a)$ is different from all $row(s)$ with $s \in S$. The set S is extended with $s_1 \cdot a$ and the observation table is extended using membership queries.

Once the inner loop terminates, the DFA that corresponds to the observation table is presented to the teacher. If the teacher accepts this DFA the algorithm terminates. If the teacher returns a counter-example, then the counter-example and all of its prefixes are added to S, the table is extended using membership queries, and the algorithm continues with the first step (the loop that verifies the observation table properties).

For example, to learn the language $L = \{(0\ 1\ 2\ 1), (0\ 2\ 2\ 2)\}$, the algorithm starts a serie of queries (shown in Table 3) until it reaches a closed and consistent observation table, Table 1. It then poses the first conjecture. The first conjecture is a DFA with only one state, that accepts no strings. After receiving the string (0 1 2 1) as (a negative) reply to the first equivalence conjecture, the algorithm poses a long sequence of membership queries (shown in Table 3) until it finally reaches a point where the observation table is again both *closed* and *consistent*, as shown in Table 2. Then the second conjecture, shown in Fig. 1, is presented to the teacher. The process could be continued until the DFA shown in Fig. 2 is reached[3] that accepts only the strings present in the target language L.

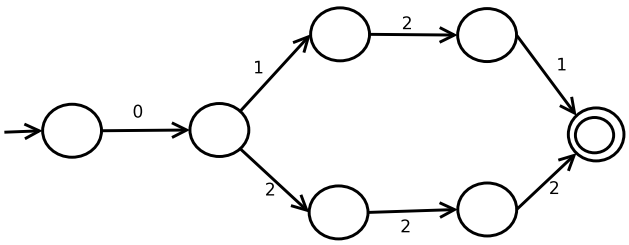

Fig. 2. Example DFA (extra state removed)

[3] The resulting DFA would have an additional non-accepting state, here omitted for clarity, where all the transitions that are not shown in the figure would converge.

Table 3. L* execution trace

L*			L* with filter
λ ? N	(0 1 2 1 2) ? N	(0 2 2 1) ? N	λ ? N
(0) ? N	(0 1 2 1 2 1) ? N	(0 1 0 2 1) ? N	(0) ? (e (0))
(1) ? N	(0 1 2 1 1 1) ? N	(2 2 1) ? N	(1) ? (s (1))
(2) ? N	(0 1 2 2 1) ? N	(1 2 1) ? N	(2) ? (s (2))
	(0 1 2 1 0 1) ? N	(0 0 2 1) ? N	
(conjecture)	(0 1 2 0 1) ? N	(0 1 2 1 2 1 2 1) ? N	(conjecture)
	(0 1 1 1) ? N	(0 1 2 1 1 1 2 1) ? N	
(0 0) ? N	(0 2 1) ? N	(0 1 2 2 1 2 1) ? N	(0 1) ? (e (0 1))
(0 1) ? N	(0 1 0 1) ? N	(0 1 2 1 0 1 2 1) ? N	(0 2) ? (e (0 2))
(0 2) ? N	(2 1) ? N	(0 1 2 0 1 2 1) ? N	(0 1 1) ? (p ((1 1)))
(0 1 0) ? N	(1 1) ? N	(0 1 1 1 2 1) ? N	(0 1 2) ? (e (0 1 2))
(0 1 1) ? N	(0 0 1) ? N	(0 2 1 2 1) ? N	(0 1 2 1) ? Y
(0 1 2) ? N	(0 1 2 1 2 2 1) ? N	(0 1 0 1 2 1) ? N	(0 1 2 2) ? (p ((1 2) (2)))
(0 1 2 0) ? N	(0 1 2 1 1 2 1) ? N	(2 1 2 1) ? N	(0 2 1) ? (p ((0 2) (1)))
(0 1 2 1) ? Y	(0 1 2 2 2 1) ? N	(1 1 2 1) ? N	(0 1 0 1 2 1) ? (p ((0) (0)))
(0 1 2 2) ? N	(0 1 2 1 0 2 1) ? N	(0 0 1 2 1) ? N	
(0 1 2 1 0) ? N	(0 1 2 0 2 1) ? N		(conjecture)
(0 1 2 1 1) ? N	(0 1 1 2 1) ? N	(conjecture)	

4 Learning DFAs Using a More Powerful Teacher

Query learning algorithms, such as L*, described in the previous section, produce a very large number of queries. This makes their use with human teachers impractical.

Depending on the target language, the number of queries of each type varies. Nevertheless, the number of membership queries is typically the dominant part in the total query count.

Membership queries have two possible answers, positive or negative. The negative answer is used to restrict the target language (with respect to the universal language containing all strings). As such, it is reasonable to expect that most languages will have a much larger count of negative membership queries than positive membership queries.

To deal with the large number of membership queries, that typically happens when learning non-trivial automata, we propose, in this work, to use a more powerful teacher. Should the answer to a membership query be negative, the teacher is requested to return additional information, namely, to identify a set of strings that would result also in negative answers to membership queries.

We consider three forms for the answer:

1. A string prefix – This form identifies the set of strings that start with the same prefix and that are also negative examples. Its use can be seen in Table 3, with the form *(s <string>)*.

2. A string suffix – The second form does the same as the first one, but with the string suffix. It identifies strings that end in the same manner and that are also negative examples. Its use can also be seen in Table 3, with the form *(e <string>)*.
3. A list of substrings – The third form can be used to specify a broader family of strings that are negative examples. Here one can identify strings by listing substrings that, when they are all present in a given string, in the same order, imply that the string is part of the described set and a negative example. For example, to identify the set of strings that contain two zeros, the reply would be the following list ((0)(0)), where (0) is a string with just one symbol, 0. Its use is also illustrated in Table 3, with the form *(p (<string1> ... <stringN>))*.

Note that these specifications can be viewed as non-deterministic finite automata (NFA).

Using the additional information, the learner can now find out the answer to a number of membership queries without making an explicit query, simply by matching the strings with the stored information using the NFA corresponding to the new answer form. This can clearly be seen in Table 3, right hand side column, where the same DFA is inferred with and without the proposed extension, resulting in an important reduction in the number of queries.

Although we are requiring more sophisticated answers from the teacher, this is a reasonable request when dealing with human teachers. A human teacher must have some informal definition of the target language in his mind, to use a query learning approach, and it is reasonable to expect that most negative answers could be justified using the proposed method. The use of a query learning method when an informal definition is already present in the human teacher's mind is necessary to obtain a minimal DFA with less effort than it would require to manually build one. As such, the extra required effort is a small one, since the human teacher would already have identified that justification in order to answer the original membership query. Moreover, in a graphical environment this could be easily implemented by allowing for the selection of parts of the query string using a mouse pointer (allowing for multiple-selection to indicate a list of substrings answer).

The proposed solution uses the L* algorithm as a "black box". A filter is placed between the teacher and the learner, which records the additional information returned by the teacher on negative membership query answers. This information is then used to reply, whenever possible, to the learner without consulting the teacher.

4.1 Example Results and Equivalence Queries

Table 4 shows the results obtained for the example DFA from Fig. 2. In this example, the strings used to reply (negatively) to the equivalence queries were (0 1 2 1), (0 2 2 2) and (2 2 2 2). Table 5 shows the distribution of membership

query answers by type and the number of answers made by the filter using the information of each type of query answer.

Even in this simple example, the number of membership queries is substantially reduced making the method usable by human teachers (L* with filter (A) in Table 4). Further results with a real application are shown in the next section.

Table 4. Query count results - simple example

	Membership Query Numbers		Equivalence Query
	Positive	Negative	Numbers
L*	2	185	4
L* with filter (A)	2	13	4
L* with filter (B)	2	13	3

(A) - Filter applied to membership queries;
(B) - Filter applied to membership and equivalence queries.

Table 5. Query counts by type - simple example

	Start with	End with	Has parts	Unjustified	Total
Teacher answers	2	4	5	2	13
Filter use counts	72	21	79	-	172

The extra information returned by the teacher could also be used to answer some equivalence queries, namely those containing strings that are not part of the target language and can be detected by the NFA already recorded. For example, the DFA in Fig. 1 admits strings, containing two or more 0s, that do not belong to the language. This could be detected as it was already stated in the end of Table 3 by "(p ((0) (0)))".

However, to detect these cases it would be necessary to obtain the product automaton between the recorded NFA and the DFA proposed by the learner, a process with quadratic complexity on the number of states. Note also that the number of states not only increases with the complexity of the language, but also with the number of extended answers (answers with the new proposed forms). This is a costly operation and would only remove, in general, a small fraction of the equivalence queries (L* with filter (B) in Table 4).

5 Application to the Inference of Document Structure

To demonstrate the use of the proposed solution, we applied it to the inference of a grammar that describes the structure of technical articles. This work is part

of an ongoing effort to automatically derive an ontology describing the structure of technical articles.

The first step was described in [15] and resulted in the segmentation of source articles into a set of symbols. These symbols are: ConferenceTitle (0), Title (1), Author (2), AbstractTitle (3), AbstractText (4), IndexTitle (6), Index (7), SectionTitle (8), SubSectionTitle (10), SubSubSectionTitle (11), SimpleText (5), FormatedText (9), FigureCaption (12).

The next step in this effort is the inference of a DFA for technical articles, using the acquired symbols. This will later enable the inference of the ontology.

To apply the approach described in this paper, we assumed that:

- The ConferenceTitle is optional;
- There can be one or more Authors;
- The Index and IndexTitle are optional;
- A section can contain some of the text elements (SimpleText, FormatedText, FigureCaption) and lower level sections.

The equivalence queries were answered using the strings in Table 6.

Table 6. Strings used in equivalence queries

Q	Strings supplied to the algorithm
1	Title Author AbstractTitle AbstractText SectionTitle SimpleText
2	Title Title Author AbstractTitle AbstractText SectionTitle SimpleText
3	Title Author AbstractTitle AbstractText IndexTitle Index SectionTitle SimpleText
4	ConferenceTitle ConferenceTitle Title Author AbstractTitle AbstractText SectionTitle SimpleText
5	Title Author AbstractTitle AbstractText SectionTitle SimpleText SubSectionTitle SimpleText SubSubSectionTitle SimpleText

Table 7 shows the number of queries resulting from the use of the L* algorithm and of the proposed solution. The resulting DFA is shown in Fig. 3. Table 8 shows the distribution of membership query answers by type and the number of answers made by the filter using the information of each type of query answer.

As the results show, the number of negative membership queries is substantially reduced. Also, at least in this example, the negative membership queries are the largest in number. This is the case for automata that have few terminal states, relatively to the total number of states, a situation that is common in real cases.

As mentioned in Sect. 4.1, the information can be used to reduce the amount of equivalence queries (L* filter (B) in Table 7), but results in only a small reduction in the number of queries.

Table 7. Query count results

	Membership Query Numbers		Equivalence Query
	Positive	Negative	Numbers
L*	99	4118	6
L* with filter (A)	99	110	6
L* with filter (B)	99	110	5

(A) - Filter applied to membership queries;
(B) - Filter applied to membership and equivalence queries.

Table 8. Query counts by type

	Start with	End with	Has parts	Unjustified	Total
Teacher answers	11	10	88	1	110
Filter use counts	101	280	3627	-	4008

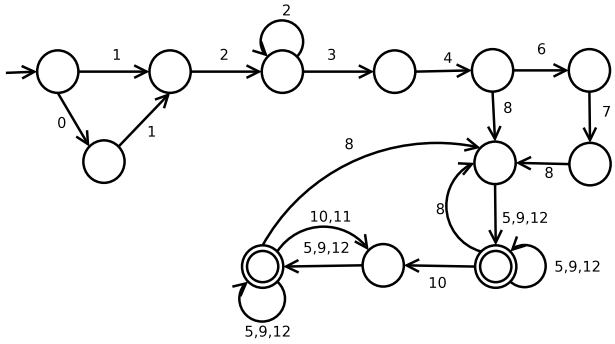

Fig. 3. DFA representing the article structure (extra state removed)

6 Conclusion

Query learning algorithms, when used with a human teacher, suffer from the excessive number of queries that are required to learn the target concept. In this work we have presented a simple extension to the well known L* algorithm that reduces this burden considerably.

The teacher is required to provide a list of sub-strings that, when present, implies that the query string does not belong to the target language. Using this extra information, many of the subsequent queries can be answered automatically by the filter. The additional information provided represents only a small amount of selection work by the user, greatly compensating for the reduction on the number of queries.

The solution is independent of the regular language query learning algorithm used, as long as the later relies mainly on membership queries. With it's use, such algorithms become a practical possibility in dealing with human teachers.

Acknowledgements

This work was partially supported by "Fundação para a Ciência e Tecnologia" under research project PSOC/EIA/58210/2004 (OntoSeaWeb-Ontology Engineering for the Semantic Web).

References

1. Gold, E.M.: Complexity of automaton identification from given data. Information and Control **37** (1978) 302–320
2. Pitt, L., Warmuth, M.: The minimum consistent DFA problem cannot be approximated within any polynomial. Journal of ACM **40** (1993) 95–142
3. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75** (1987) 86–106
4. Gold, E.M.: System identification via state characterization. Automatica **8** (1972) 621–636
5. Schapire, R.E.: The Design and Analysis of Efficient Learning Algorithms. MIT Press, Cambridge, MA (1992)
6. Nevill-Manning, C., Witten, I.H., Maulsby, D.L.: Modeling sequences using grammars and automata. In: Proceedings Canadian Machine Learning Workshop. (1994) 15–18
7. Hsu, C.N., Dung, M.T.: Generating finite-state transducers for semi-structured data extraction from the web. Information Systems **23** (1998) 521–538
8. Witten, I.H.: Adaptive text mining: inferring structure from sequences. Journal of Discrete Algorithms **2** (2004) 137–159
9. Laender, A.H.F., Ribeiro-Neto, B.A., da Silva, A.S., Teixeira, J.S.: A brief survey of web data extraction tools. SIGMOD Record **31** (2002) 84–93
10. Ribeiro-Neto, B.A., Laender, A.H.F., da Silva, A.S.: Extracting semi-structured data through examples. In: Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management, ACM (1999) 94–101
11. Adelberg, B.: NoDoSE - a tool for semi-automatically extracting semi-structured data from text documents. In: Proceedings ACM SIGMOD International Conference on Management of Data. (1998) 283–294
12. Califf, M.E., Mooney, R.J.: Relational learning of pattern-match rules for information extraction. In: Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence. (1999) 328–334
13. Soderland, S.: Learning information extraction rules for semi-structured and free text. Machine Learning **34** (1999) 233–272
14. Angluin, D.: Queries and concept learning. Machine Learning **2** (1988) 319–342
15. Martins, A.L., Pinto, H.S., Oliveira, A.L.: Towards automatic learning of a structure ontology for technical articles. In: Semantic Web Workshop at SIGIR 2004. (2004)

Bibliography

[1] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[2] André Luís Martins, H. Sofia Pinto, and Arlindo L. Oliveira. Towards Automatic Learning of a Structure Ontology for Technical Articles. In *Semantic Web Workshop at SIGIR 2004*, 2004.

[3] André Luís Martins, H. Sofia Pinto, and Arlindo L. Oliveira. Using a More Powerful Teacher to Reduce the Number of Queries of the L* Algorithm in Practical Applications. In C. Bento, A. Cardoso, and G. Dias, editors, *Proceedings of EPIA 2005*, volume 3808 of *Lecture Notes in Artificial Intelligence*, pages 325–336. Springer-Verlag, 2005.

[4] Betsy Humphreys, Donald Lindberg, Harold Schoolman, and G. Octo Barnett. The Unified Medical Language System: An Informatics Research Collaboration. *Journal of the American Medical Informatics Association*, 5(1):1–11, 1998.

[5] James F. Allen. Maintaining Knowledge About Temporal Intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[6] IEEE. Standard Upper Ontology. http://suo.ieee.org, (last accessed Dec 2005).

[7] John F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks Cole Publishing Co., 1999.

[8] Stanford University. Knowledge Interchange Format. http://www-ksl.stanford.edu/knowledge-sharing/kif/, (last accessed Dec 2005).

[9] W3C. OWL. http://www.w3.org/2004/OWL/, (last accessed Dec 2005).

[10] Richmond H. Thomason and John F. Horty. Logics for Inheritance Ttheory. In M. Reinfrank, J. de Kleer, M. Ginsberg, and E. Sandewall, editors, *Proceedings of the 2nd International Workshop on Non-monotonic Reasoning*, volume 346 of *Lecture Notes in Artificial Intelligence*, pages 220–237. Springer-Verlag, 1989.

[11] Achille Varzi. Stanford Encyclopedia of Philosophy - Mereology. http://plato.stanford.edu/entries/mereology/, (last accessed Dec 2005).

[12] Alexander Maedche and Steffen Staab. *Ontology Learning for the Semantic Web*. Kluwer Academic Publishers, 2002.

[13] Mariano Fernandez, Asuncion Gomez-Perez, and Natalia Juristo. METHONTOLOGY: from Ontological Art Towards Ontological Engineering. In *Proceedings of the AAAI 97 Spring Symposium Series on Ontological Engineering*, pages 33–40. The AAAI Press, 1997.

[14] Mike Uschold and Martin King. Towards a Methodology for Building Ontologies. In *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing held in conduction with IJCAI 1995*, 1995.

[15] Michael Grüninger and Mark. S. Fox. Methodology for the Design and Evaluation of Ontologies. In *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing held in conduction with IJCAI 1995*, 1995.

[16] Mariano Fernández López. OntoWeb Deliverable 1.4: A Survey on Methodologies for Developing, Maintaining, Evaluating and Reengineering Ontologies. http://ontoweb.org/Members/huro/MyPublications/OntoWeb%20Deliverable%201.4, 2002.

[17] Natalya Fridman Noy and Carole D. Hafner. The State of the Art in Ontology Design. *AI Magazine*, 18(3):53–74, 1997.

[18] Asunción Gómez-Pérez. Evaluation of taxonomic knowledge in ontologies and knowledge bases. In *Proceedings of the 12th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW 1999)*, volume 2, pages 6.1.1–6.1.18. SRDG Publication, University of Calgary, 1999.

[19] Nicola Guarino and Christopher Welty. Evaluating Ontological Decisions with OntoClean. *Communications of the ACM*, 45(2):61–65, 2002.

[20] Andreas Faatz and Ralf Steinmetz. Ontology Enrichment with Texts from the WWW. In *Proceedings of 2nd Workshop on Semantic Web Mining 2002, ECML 2002*, pages 20–34, 2002.

[21] Bruno Bachimont, Antoine Isaac, and Raphaël Troncy. Semantic Commitment for Designing Ontologies: A Proposal. In A. Gomez-Perez and V.R. Benjamins, editors,

Proceedings of 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW 2002), volume 2473 of *Lecture Notes in Artificial Intelligence*, pages 114–121. Springer-Verlag, 2002.

[22] Michele Missikoff, Roberto Navigli, and Paola Velardi. The Usable Ontology: An Environment for Building and Assessing a Domain Ontology. In Ian Horrocks and James A. Hendler, editors, *International Semantic Web Conference*, volume 2342 of *Lecture Notes in Computer Science*, pages 39–53. Springer-Verlag, 2002.

[23] Latifur Khan and Feng Luo. Ontology Construction for Information Selection. In *Proceedings of 14th IEEE International Conference on Tools with Artificial Intelligence*, pages 122–127. IEEE Computer Society, 2002.

[24] Chung Hee Hwang. Incompletely and Imprecisely Speaking: Using Dynamic Ontologies for Representing and Retrieving Information. In *Proceedings of the 6th International Workshop on Knowledge Representation meets Databases (KRDB 1999)*, volume 21 of *CEUR Workshop Proceedings*, pages 14–20. CEUR-WS.org, 1999.

[25] Eneko Agirre, Olatz Ansa, Eduard Hovy, and David Martínez. Enriching Very Large Ontologies using the WWW. In *Proceedings of the Ontology Learning Workshop of ECAI-2000*, volume 31 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2000.

[26] Joaquín Dopazo and José Maria Carazo. Phylogenetic Reconstruction Using an Unsupervised Growing Neural Network That Adopts the Topology of a Phylogenetic Tree. *Journal of Molecular Evolution*, 44(2):226–233, 1997.

[27] Enrique Alfonseca and Suresh Manandhar. An Unsupervised Method for General Named Entity Recognition and Automated Concept Discovery. In *Proceedings of the 1st International WordNet Conference*, 2002.

[28] Enrique Alfonseca and Suresh Manandhar. Extending a Lexical Ontology by a Combination of Distributional Semantics Signatures. In A. Gomez-Perez and V.R. Benjamins, editors, *Proceedings of 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW 2002)*, volume 2473 of *Lecture Notes in Artificial Intelligence*, pages 1–7. Springer-Verlag, 2002.

[29] Enrique Alfonseca and Suresh Manandhar. Improving an Ontology Refinement Method with Hyponymy Patterns. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002)*. ELDA/ELRA, 2002.

[30] Marti A. Hearst. Automatic Acquisition of Hyponyms from Large Text Corpora. In *Proceedings of the 14th Conference on Computational linguistics*, pages 539–545. Association for Computational Linguistics, 1992.

[31] Pavel Makagonov, Alejandro Ruiz Figueroa, Konstantin Sboychakov, and Alexander Gelbukh. Learning a Domain Ontology from Hierarchically Structured Texts. In *Proceedings of the Workshop Learning and Extending Lexical Ontologies by Using Machine Learning Methods, ICML 2005*, 2005.

[32] Udo Hahn and Klemens Schnattinger. Towards Text Knowledge Engineering. In *Proceedings 15th National Conference on Artificial Intelligence (AAAI 1998)*, pages 524–531. The AAAI Press, 1998.

[33] Deryl Lonsdale, Alan Melby, Yihong Ding, and David Embley. Peppering Knowledge Sources with SALT: Boosting Conceptual Content for Ontology Generation. In *Proceedings of the Workshop on Semantic Web Meets Language Resources of AAAI 2002*, pages 30–36. The AAAI Press, 2002.

[34] Joerg-Uwe Kietz, Alexander Maedche, and Raphael Volz. A Method for Semi-Automatic Ontology Acquisition from a Corporate Intranet. In *Proceedings of Ontologies and Text Workshop of EKAW 2000*, volume 51 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2000.

[35] Gerd Stumme and Alexander Maedche. FCA-Merge: Bottom-up Merging of Ontologies. In *7th International Conference on Artificial Intelligence (IJCAI 2001)*, pages 225–230. Morgan Kaufmann, 2001.

[36] Natalya Fridman Noy and Mark A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence (AAAI 2000)*, pages 450–455. The AAAI Press / The MIT Press, 2000.

[37] Deborah L. McGuinness, Richard Fikes, James Rice, and Steve Wilder. An environment for merging and testing large ontologies. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, pages 483–493. Morgan Kaufmann, 2000.

[38] Alexandre Delteil, Catherine Faron-Zucker, and Rose Dieng. Learning Ontologies from RDF Annotations. In *Proceedings of the Workshop on Ontology Learning OL-2001 of IJCAI 2001*, volume 38 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2001.

[39] Christos Papatheodorou, Alexandra Vassiliou, and Bernd Simon. Discovery of Ontologies for Learning Resources using Word-based Clustering. In *Proceedings of EDMEDIA 2002*, volume 2002, pages 1523–1538. AACE, 2002.

[40] Mike Perkowitz and Oren Etzioni. Adaptive Web Sites: Automatically Synthesizing Web Pages. In *Proceedings 15th National Conference on Artificial Intelligence (AAAI 1998)*, pages 727–732. The AAAI Press, 1998.

[41] AnHai Doan, Pedro Domingos, and Alon Levy. Learning Source Descriptions for Data Integration. In *Proceedings of the International Workshop on The Web and Databases (WebDB)*, pages 81–86, 2000.

[42] Raphael Volz, Daniel Oberle, Steffen Staab, and Rudi Studer. OntoLiFT Prototype. IST Project 2001-33052 WonderWeb Deliverable 11, 2003.

[43] Makoto Murata, Dongwon Lee, and Murali Mani. Taxonomy of XML Schema Languages using Formal Language Theory. In *Proceedings of Extreme Markup Languages 2001*. IDEAlliance, http://www.mulberrytech.com/Extreme/Proceedings/site.html, 2001.

[44] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. A Brief Survey of Web Data Extraction Tools. *SIGMOD Rec.*, 31(2):84–93, 2002.

[45] W3C. The Document Object Model. http://www.w3.org/DOM, (last accessed Dec 2005).

[46] Davi de Castro Reis, Robson Braga Araújo, Altigran Soares da Silva, and Berthier A. Ribeiro-Neto. A Framework for Generating Attribute Extractors for Web Data Sources. In *Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002)*, pages 210–226. Springer-Verlag, 2002.

[47] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 109–118. Morgan Kaufmann Publishers, 2001.

[48] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *Proceedings of the 16th International Conference on Data Engineering (ICDE 2000)*, pages 611–621. IEEE Computer Society, 2000.

[49] Arnaud Sahuguet and Fabien Azavant. Building Intelligent Web Applications using Lightweight Wrappers. *Data Knowledge Engineering*, 36(3):283–316, 2001.

[50] Noam Chomsky. Three models for the Description of Language. In *Institute of Radio Engineers Transactions on Information Theory 2*, pages 113–124. 1956.

[51] John E Hopcroft and Jeffrey D Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[52] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[53] Dana Angluin. Inference of Reversible Languages. *Journal of ACM*, 29(3):741–765, 1982.

[54] Yasubumi Sakakibara. Efficient Learning of Context-free Grammars from Positive Structural Examples. *Information and Computation*, 97(1):23–60, 1992.

[55] Yasubumi Sakakibara and Mitsuhiro Kondo. GA-based Learning of Context-Free Grammars using Tabular Representations. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999)*, pages 354–360. Morgan Kaufmann Publishers, 1999.

[56] E. M. Gold. System Identification Via State Characterization. *Automatica*, 8:621–636, 1972.

[57] Dana Angluin. Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, 75(2):86–106, 1987.

[58] Ronald L. Rivest and Robert E. Schapire. Inference of Finite Automata using Homing Sequences. *Information and Control*, 103(2):299–347, 1993.

[59] Rajesh Parekh and Vasant Honavar. An Incremental Interactive Algorithm for Regular Grammar Inference. In L. Miclet and C. Higuera, editors, *Proceedings of the Third International Colloquium on Grammar Inference (ICGI 1996)*, volume 1147 of *Lecture Notes in Computer Science*, pages 238–250. Springer-Verlag, 1996.

[60] Orlando Cicchello and Stefan C. Kremer. Inducing Grammars from Sparse Data Sets: A Survey of Algorithms and Results. *Journal of Machine Learning Research*, 4:603–632, 2003.

[61] Miguel Bugalho and Arlindo L. Oliveira. Inference of Regular Languages using State Merging Algorithms with Search. *Pattern Recognition*, 38(9):1457–1467, 2005.

[62] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

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

[64] J. Rissanen. Modeling by the Shortest Data Description. *Automatica*, 14:465–471, 1978.

[65] J. Rissanen. A Universal Prior for Integers and Estimation by Minimum Description Length. *Annals of Statistic*, 11:416–431, 1983.

[66] Peter D. Grünwald, In Jae Myung, and Mark A. Pitt. *Advances in Minimal Description Length: Theory and Applications*. The MIT Press, 2005.

[67] J. Ross Quinlan and Ronald L. Rivest. Inferring Decision Trees Wsing the Minimum Description Length Principle. *Information and Computation*, 80(3):227–248, 1989.

[68] Arlindo L. Oliveira and A. Sangiovanni-Vincentelli. Using the Minimum Description Length Principle to Infer Reduced Ordered Decision Graphs. *Machine Learning*, 25(1):23–50, 1996.

[69] Peter D. Grünwald. A Minimum Description Length Approach To Grammar Inference. In *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, volume 1040 of *Lecture Notes in Artificial Intelligence*, pages 203–216. Springer-Verlag, 1996.

[70] Yu M. Shtarkov. Universal Sequential Coding of Single Messages. *Problems of Information Transmission*, 23(3):3–17, 1987.

[71] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm. In G. Slutzki V. Honavar, editor, *Proceedings of the Fourth International Colloquium on Grammatical Inference (ICGI 1998)*, volume 1433 of *Lecture Notes in Computer Science*, pages 1–12. Springer-Verlag, 1998.

[72] M. Tomita. Dynamic Construction of Finite-State Automata From Examples Using Hill-Climbing. In *Proceedings of Fourth Annual Cognitive Science Conference*, pages 105–108, 1982.

[73] Alfred Aho and Thomas Peterson. A Minimum Distance Error-Correcting Parser for Context-Free Languages. *SIAM Journal of Computation*, 1(4):305–312, December 1972.