

# Inference of Concise DTDs from XML Data

Geert Jan Bex Frank Neven  
Hasselt University and  
Transnational University of Limburg

Thomas Schwentick  
Dortmund University

Karl Tuyls  
Maastricht University and  
Transnational University of Limburg

## ABSTRACT

We consider the problem to infer a concise Document Type Definition (DTD) for a given set of XML-documents, a problem which basically reduces to learning of *concise* regular expressions from positive example strings. We identify two such classes: single occurrence regular expressions (SOREs) and chain regular expressions (CHAREs). Both classes capture the far majority of the regular expressions occurring in practical DTDs and are succinct by definition. We present the algorithm iDTD (infer DTD) that learns SOREs from strings by first inferring an automaton by known techniques and then translating that automaton to a corresponding SORE, possibly by repairing the automaton when no equivalent SORE can be found. In the process, we introduce a novel automaton to regular expression rewrite technique which is of independent interest. We show that iDTD outperforms existing systems in accuracy, conciseness and speed. In a scenario where only a very small amount of XML data is available, for instance when generated by Web service requests or by answers to queries, iDTD produces regular expressions which are too specific. Therefore, we introduce a novel learning algorithm CRX that directly infers CHAREs (which form a subclass of SOREs) without going through an automaton representation. We show that CRX performs very well within its target class on very small data sets. Finally, we discuss incremental computation, noise, numerical predicates, and the generation of XML Schemas.

## 1. INTRODUCTION

### 1.1 Motivation

XML is the lingua franca for data exchange on the Internet [2]. Within applications or communities, XML data is usually not arbitrary but adheres to some structure possibly imposed by a schema. The advantages offered by the presence of such a schema are numerous. The most direct application is of course automatic validation of the document structure. Input validation, for instance, not only facilitates automatic processing but also ensures soundness of the in-

put data. Indeed, unvalidated input from web requests is considered as the number one vulnerability for web applications [1]. The presence of a schema allows for automation and optimization of search, integration, and processing of XML data (cf., e.g., [7, 17, 30, 31, 38, 50]). Various software development tools such as Castor<sup>1</sup> and SUN's JAXB<sup>2</sup> rely on schemas to perform object-relational mappings for persistence. Further, the existence of schemas is imperative when integrating (meta) data through schema matching [43] and in the area of generic model management [8, 32]. A final advantage of a schema is that it assigns meaning to the data. That is, it provides a user with a concrete semantics of the document and aids in the specification of meaningful queries over XML data. Although the examples mentioned here just scrape the surface of current applications, they already underscore the importance of schemas accompanying XML data.

Unfortunately, in spite of the above mentioned advantages, a schema is not mandatory and many XML-documents do not possess one. Indeed, in a recent study, Barbosa et al. [6, 33] have shown that approximately half of the XML documents available on the web do not refer to a schema. In another study Bex et al. [9, 10] noted that about two-thirds of XSDs gathered from schema repositories and from the web are not valid with respect to the W3C XML Schema specification [48], rendering them essentially useless for immediate application. A similar observation was noted by Sahuguet [44] concerning DTDs.

Based on the above described advantages and the lack of schemas in practice, it is essential to devise algorithms that can infer a schema for a given collection of XML documents when none, or no syntactic correct one, is present. The latter problem is also acknowledged by Florescu [22] who emphasizes that in the context of data integration “*We need to extract good-quality schemas automatically from existing data and perform incremental maintenance of the generated schemas*”. In this paper, we describe two novel schema inference algorithms outperforming existing systems in accuracy, conciseness and speed.

Before we outline our approach, we give two applications of schema inference in situations where a schema is already available: schema cleaning and dealing with noise. Even when schemas do exist, it can be advantageous to derive one solely from the XML data at hand. Indeed, sometimes schemas can be too general with respect to the data they are to represent. This is, for instance, nicely illustrated by

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '06, September 12-15, 2006, Seoul, Korea.

Copyright 2006 VLDB Endowment, ACM 1-59593-385-9/06/09.

<sup>1</sup><http://www.castor.org/>

<sup>2</sup><http://java.sun.com/webservices/jaxb/>

the following real-world example taken from the Protein Sequence Database DTD [34]. Consider the definition of the `refinfo`-element:

```
authors,citation,volume?,month?,
year,pages?,(title|description)?,xrefs?
```

An analysis of the available XML corpus (683 megabyte of data) shows that the `refinfo`-element is better described by the following which is more strict than the original:

```
authors,citation,(volume|month),
year,pages?,(title|description)?,xrefs?
```

The latter emphasizes that `volume` and `month` do not occur together: one either specifies a journal articles month of publication, or the volume it appeared in, but not both. The latter definition for `refinfo` is derived by our algorithms, illustrating that they can be used to better understand the semantics of the data and adapt the schema when necessary. In general, schema inference can be used to restrict schemas to the relevant subset that is needed by the application at hand, thereby facilitating difficult tasks like schema matching and data integration. Indeed, as argued by Hinkelman [28], industry-level standards are generally too loosely defined, which can result in XML schemas in which many business structures are formally specified as being optional.

A final example illustrating the importance of schema inference is one in the context of noisy data. We investigated a corpus of XHTML documents gathered from the web and found that an astonishing 89 % of 2092 documents was not valid to the XHTML Transitional specification<sup>3</sup>. Inference of a schema from the data at hand and comparing it with the schema provided by the specification provides a uniform view of the kind of errors. Further, given that one often has no choice but to deal with such noisy data one may derive a schema from a subset and work with that rather than with the official specification to retain at least a minimal validation. We refer to Section 9 for a more thorough discussion.

## 1.2 Problem setting

Given a collection of XML documents, a schema should be inferred in an automatic way without intervention of the user and can therefore solely be based on the XML data at hand. Like for any effective inference algorithm, the generated schema should strike a good balance between (1) specialization, i.e., covering all XML documents in the sample in a minimal way; and, (2) generalization, i.e., covering all documents satisfying the target schema but which are not necessarily present in the sample.

In this respect, schema inference problems come in two flavors. First, there is the setting when only little XML data is present, for instance, when XML is returned as answers to queries or Web service requests [39, 40]. In such a case, a schema inference algorithm should balance more towards generalization than to specificity as it is unlikely that a rich class of schemas can be learned from only a limited number of data instances. The other scenario is one in which there is a huge amount of XML data available, for instance, when the data resides in a native XML databases or is generated in bulk from existing (say relational) data. In this case, there usually is enough information to derive a highly specific schema in a rich class and a learning algorithm should therefore favor specialization over generalization.

We consider the inference of concise Document Type Definitions (DTDs) in both of the above settings. We briefly

<sup>3</sup><http://www.w3.org/TR/xhtml1/>

motivate why we want to infer DTDs and not the much richer formalism of XML Schema [49]. The most important reason is that, as we explain in the related work section, DTD inference has not adequately been addressed yet: existing systems do not perform well when tested on real world or sparse XML data, and as a result, generate lengthy and long-winded regular expressions. Secondly, DTD inference is a subcase of XSD inference. Indeed, recent characterizations by Bex et al. [9] show that the structural core of XML Schema, that is, the pure sets of trees which are definable by XSDs, corresponds to DTDs extended with vertical regular expressions. Therefore, one cannot hope to successfully infer XSDs without good algorithms for the derivation of DTDs. So, the present work can be seen as a first necessary step towards XML Schema inference.

As DTDs can be abstracted by context-free grammars with regular expressions (REs) at their right-hand sides, DTD inference reduces to learning of REs from positive example strings. Indeed, for every element name we need to infer an RE describing all the strings occurring below that element name in the XML corpus. Unfortunately, a seminal result by Gold [25] shows that the class of *all* REs cannot be learned from positive data only. This means that no matter how many example strings from the target language are provided, no algorithm can infer every target RE. As the framework for schema inference from XML data is exactly such that only positive example strings are provided, it is unrealistic to develop inference algorithms for the class of all DTDs. One of the main challenges is therefore to identify subclasses of REs which include the far majority of REs occurring in practical DTDs, which are concise, and which can be learned efficiently from positive data only.

We present two such classes:

1. The class of **single occurrence REs (SOREs)**, these are REs in which every element name can occur at most once. For instance,  $((b?(a+c))^+d)^+e$  is SORE while  $a(a+b)^*$  is not as  $a$  occurs twice.
2. The class of **chain regular expressions (CHAREs)** which are those SOREs consisting of a sequence of factors  $f_1 \cdots f_n$  where every factor is an expression of the form  $(a_1 + \cdots + a_k)$ ,  $(a_1 + \cdots + a_k)?$ ,  $(a_1 + \cdots + a_k)^+$ , or,  $(a_1 + \cdots + a_k)^*$ , where  $k \geq 1$  and every  $a_i$  is an alphabet symbol. For instance, the expression  $a(b+c)^*d^+(e+f)?$  is a CHARE, while  $(ab+c)^*$  and  $(a^*+b)^*$  are not.

Note that every SORE, and therefore, also every CHARE is deterministic (or one-unambiguous [12]) as required by the XML specification.

These classes certainly satisfy the relevance criteria mentioned above: an examination of the 819 DTDs and XSDs gathered from the Cover Pages<sup>4</sup> (including many high-quality XML standards) as well as from the web at large, reveals that more than 99% of the REs occurring in practical schema's are CHAREs (and therefore also SOREs) [10]. Furthermore, they are succinct by definition: every element name can occur only once. Hence, the size of the generated RE is always linear in the number of different element names occurring in the corpus.

In this paper, we show that these classes of REs can be efficiently learned, thereby deriving efficient algorithms for

<sup>4</sup><http://xml.coverpages.org/>



















