Grammar Inference for Web Documents

Shahab Kamali
David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, ON, Canada
skamali@cs.uwaterloo.ca

Frank Wm. Tompa
David R. Cheriton School of Computer Science
University of Waterloo, Waterloo, ON, Canada
fwtompa@cs.uwaterloo.ca

ABSTRACT

Presentational XML documents, such as XHTML or Presentation MathML, use XML tags mainly for formating purposes, while descriptive XML applications, such as a well-structured movie database, use tags to structure data items in a semantically meaningful way. There is little semantic connection between tags in a presentational XML document and its content, so the tagging is often complex and seemingly ambiguous. These differences make inference of the underlying structure more difficult for presentational XML.

The problem of schema or grammar inference has been studied mostly for descriptive XML, and proposed solutions are often ineffective for presentational XML. On the other hand, there are many applications such as data extraction tools and special-purpose search engines that need to infer structure from presentational XML. Current proposals for such systems provide only partial solutions to this problem.

Restrictions imposed by DTDs and XML Schemas make them insufficient to describe many presentational XML documents effectively. In this paper we use regular tree grammars to define a class of grammars that is able to model many published presentational XML documents. We also propose an algorithm to infer such grammars, and prove that we can infer an appropriate grammar with high probability from given samples. We also empirically evaluate our algorithm by applying it to various types of presentational XML and comparing it to other algorithms.

1. INTRODUCTION

Although XML was designed primarily to support descriptive markup for text, many XML documents employ extensive presentational tags. For example, an XHTML page generally contains an arbitrary mixture of forms, tables, images, and blocks of text with various font styles, often separated into untyped logical divisions. Whereas descriptive XML encodes data records and is used to exchange data between applications or to store structured text in a database, presentational XML is mostly used to format documents in order to publish them. Unlike descriptive XML, presentational XML usually has a less explicit structure, and elements representing document features appear as needed in a given document. Moreover, because tags are used mostly for formatting, there is no strong semantic relationship between tags and their text contents. Thus, similar tagging conventions can be used arbitrarily often in various semantically distinct parts of a document and identical tags might represent diverse entity types.

An example of presentational XML is shown in Figure 1, where university faculties are listed in the body of an XHTML document. There is no explicit delimiter that separates universities and their contents from each other and from the rest of the document. Moreover, some fields appear often and in various locations within a listing (e.g. `<br/>` delimits disparate entities). Figure 1 also illustrates an alternative representative XML for the university listings. Tags represent the semantics of their contents, the structure is cleaner, and there are fewer tags than are included in the presentational XML. Note that each university listing contains a list of faculties of arbitrary length, some faculty entries have arbitrary sublists, and some fields appear in only some of the listings, e.g. the subtext between the university name and the list of faculties appears for the second university only. The presence of nested lists and optional elements and the absence of clear borders between records in presentational XML make the inference of a grammar difficult.

```
<b>University of Kitcherloo</b>
  <br/>
  <i>Faculties</i>
    <ul>
      <li>Science</li>
      <li>Humanities
        <div class="depts">
          &ndash; French
        </div>
        <div class="depts">
           &ndash; English
        </div>
      </li>
      <li>Social Sciences</li>
   </ul>
  <i>admin@kloo.ca</i>
<br/>
<b>Outerloo College</b>
  <br/>
  (north campus)
  <br/>
  <i>Faculties</i>
    <ul>
      <li>Photography</li>
      <li>Medical Supplies</li>
   ...
```

```
<university>
  <name>
    University of Kitcherloo
  </name>
  <faculties>
    <faculty>Science</faculty>
    <faculty>Humanities
      <depts>
        <dept>French</dept>
        <dept>English</dept>
      </depts>
    </faculty>
    <faculty>
      Social Sciences
    </faculty>
  </faculties>
  <email>admin@kloo.ca</email>
</university>
<university>
  <name>Outerloo College</name>
  <loc>north campus</loc>
  <faculties>
    <faculty>Photography</faculty>
    <faculty>
      Medical Supplies
    </faculty> ...
```

Figure 1: Presentational XML (left) vs. descriptive XML (right)

Fourteenth International Workshop on the Web and Databases (WebDB 2011), June 12, 2011 Athens, Greece.
.

Schema induction is mostly studied in the context of descriptive XML, aiming to infer appropriate DTDs or XSDs for a set of sample XML documents. XTRACT [8] is a system for inferring DTDs that, given a set of sample XML documents, first builds a candidate set of regular expressions (REs) using some heuristics, then defines a cost function to choose a subset of such candidates, and finally combines them to produce a DTD. The proposed heuristics can generate rather simple REs only, e.g. optional components nested within repetitive patterns are not allowed. Furthermore, experiments on real data show that XTRACT generates long-winded regular expressions that are difficult to understand [2]. Bex et al. propose algorithms for identifying regular expressions to infer DTDs [2] or XSDs [1] from XML documents. Their algorithm can infer rather simple grammars such as single occurrence REs (i.e. each non-terminal appears at most once in a RE). Their extended algorithm for inferring k-occurrence REs (i.e. each non-terminal appears at most k times in a RE) is computationally very expensive and requires huge sample sets to infer a grammar correctly (e.g. thousands of samples for medium-size grammars).

For many extraction tasks based on presentational XML, the number of samples is very limited [6, 12, 13], and in some cases it is assumed that only one sample is provided [7, 12, 13]. Furthermore descriptive schema induction algorithms often rely on some restrictions imposed by specific schema languages. For example competing non-terminals (i.e. non-terminals that have the same terminals in their production rules, e.g. $A \to tr_1$ and $B \to tr_2$) cannot occur anywhere in a DTD, and XSLs do not allow competing non-terminals within a production rule [14]. This is clearly not adequate in the case of presentational XML where, for example, two sibling nodes in an XHTML document with label `<div>` might represent two completely different types.

For presentational XML on the other hand, there has not been much effort to develop a general schema inference framework, although the existence of a schema can greatly improve many systems that mine such data in order to extract information or to query them. Examples include extracting data records from a single web-page [7, 12, 13], generating wrappers for a set of web pages (also called wrapper induction) [6, 17], and classifying semistructured objects (e.g. polynomials vs. matrices encoded with MathML). Current proposals for such systems mostly rely on domain-specific heuristics that cannot be effectively applied to other similar scenarios. Moreover such approaches make simplifying assumptions about the structure of presentational XML that affect their robustness significantly. For example some approaches consider only single occurrence grammars [13, 17], some do not allow optionals and disjunctive expressions within repetitive patterns [12, 15, 16], and some do not allow nested repeating patterns [4, 12, 16].

In this paper we propose a grammar induction framework for presentational XML. Our contributions are as follows:

- We define a class of regular tree grammars, called *k-normalized regular tree grammars* (k-NRTGs), that can model most presentational XMLs on the Web.
- We propose an inference algorithm and prove that even if a small number of samples are provided, the correct grammar is inferred with high probability.

Note that the algorithms in this paper can also be applied to descriptive XML documents, but we will concentrate on the more difficult problem of presentational XML.

The rest of this paper is organized as follows. We define k-normalized regular tree grammars in Section 2. In Section 3 we explain an inference algorithm based on k-normalized regular tree grammars, and we prove that it can infer the correct grammar with high probability. We finally present an evaluation of our algorithm before concluding the paper.

2. DEFINITIONS

Whereas a DTD describes valid tagged text in a document and XML Schema describes valid elements and attributes in a corresponding Infoset model, a regular tree grammar (the basis of RELAX NG) describes valid parse trees [5, 14]. These grammars describe a "language" of parse trees in terms of regular expressions, just as a conventional grammar uses regular expressions on the right sides of its productions to describe a language of strings. This makes regular tree grammars particularly suited to our grammar inference task.

DEFINITION 1. *A regular tree grammar is a 4-tuple $H = (N, T, S, P)$ where N is a finite set of non-terminals, T is a finite set of terminals, S is a set of start symbols ($S \subseteq N$), and P is a set of production rules of the form $X \to aR$ where X is a non-terminal, a is a terminal, and R is either empty or of the form $[r]$ where r is a regular expression over N.*

Terminals correspond to labels for a tree's nodes, and they cannot be null. Each non-terminal X, with production rule $X \to a[r]$, generates a set of trees, where a represents the label of the root, and r is the content model of the production rule, which may be empty. Different non-terminals cannot generate the same sets of trees. We allow the following operators within a regular expression: *concatenation*, *disjunction*, and *bound repetition*. Concatenation of E_1 and E_2 is represented as E_1E_2 and their disjunction is represented as $E_1|E_2$. *Bound repetition* is represented as $E^{[l,u]}$, where E is a regular expression that is constrained to repeat at least l times and at most u times, and $0 < l \leq u$. For example $E^{[1,3]}$ means E should occur at least once and at most three times. For convenience, we define $E^k = E^{[k,k]}$, $E? = E|\epsilon$ (i.e., optional, where ϵ denotes the empty tree), $E^+ = E^{[1,\infty]}$, and $E^* = E^+|\epsilon$. The language of a regular expression E is the set of all strings generated by E and is denoted by $L(E)$. Two regular expressions are equivalent if they generate the same language. An *atomic* regular expression is a non-terminal or an expression of the form E^+ or $(E_1|E_2|...|E_k)$ where $k > 1$ and each E_i is an atomic regular expression (i.e. a limited concatenation of expressions is non-atomic). A regular expression E is in disjunctive form if $E = (E_1|E_2|...|E_k)$, $k > 1$, and $L(E_i) \neq L(E_j)$ if $i \neq j$. (Note that by definition $E?$ and E^* are disjunctive regular expressions when $E \neq \epsilon$.)

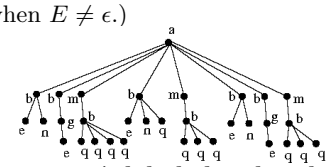

Figure 2: A labeled ordered tree.

As an example consider the tree in Figure 2, which can be generated by the following regular tree grammar:

$H = \{N, T, S, P\}$, $T = \{a, b, e, g, m, n, q\}$, $N = \{E, N, Q, G, I, J, K, L, M\}$, $S = \{L\}$

$P = \{E \to e, N \to n, Q \to q, G \to g[E], J \to b[G], I \to b[ENQ?], K \to b[Q^+], M \to m[K], L \to a[(IJ?M)^+]\}$

Note that I, J, and K share a common root symbol. This grammar cannot be described by a DTD or XSD.

We wish to compare subtrees by size (i.e., the number of nodes they contain), and thus we assign *weights* to each nonterminal N in the grammar to represent the maximum size of a tree that can be generated by N or a constant W if this size is unbounded or it exceeds W. We augment this by assigning weights to regular expressions as follows:

if E is a nonterminal N where $N \rightarrow n[E']$,
$\omega(E) = min\{W, 1 + \omega(E')\}$
$\omega(E_1E_2...E_k) = min\{W, \sum \omega(E_i)\}$
$\omega(E^{[l,u]}) = min\{W, u.\omega(E)\}$
$\omega(E_1|E_2|...|E_k) = max\{\omega(E_i)\}$

Alternative monotonic weighting schemes could be adopted instead, provided they assign the same (positive) weights to equivalent expressions.

Definition 2. *For an atomic regular expression A, let $\kappa(A) = 1$ if either A is disjunctive or $\epsilon \in L(A)$ and $\kappa(A) = 0$ otherwise, and let α be a constant such that $0 \leq \alpha < 1$. A regular expression of the form $R = A_1A_2 \dots A_n$, where A_is are atomic regular expressions, is* repeatable *if $V(R) = \frac{\sum \kappa(A_i)\omega(A_i)}{\sum \omega(A_i)} \leq \alpha$. A regular expression E is* a repetitive pattern *if $E = (R)^{[l,u]}$ and $u > 1$. E is a valid repetitive pattern if R is valid and repeatable. A regular expression is* valid *if it contains only valid repetitive patterns or no repetitive pattern at all.*

For example if $E = (A?B(C|D))^+$, $\alpha = 0.5$, and $\omega(A) = \omega(B) = \omega(C) = \omega(D) = 5$, then $V(E) = \frac{10}{15} > \alpha$, so E is not a valid repetitive pattern. This reflects the fact that the production includes "too few" terms that must appear in every repetition. On the other hand, if A and $(C|D)$ are "small," say $\omega(A) = \omega(C) = \omega(D) = 1$, then $V(E) = \frac{2}{7}$ which is less than α, and therefore E would be a valid repetitive pattern: the "large" B appears in every repetition. If $\alpha = 0$ then no disjunctive or optional components are allowed in a repetitive pattern.

Definition 3. *Let $U = U_1U_2...U_m$ and $V = V_1V_2...V_n$ be two regular expressions in which every U_i and V_i is atomic. The* alignment *of U and V is a maximal ordered sequence $< (U_{i_1}, V_{j_1}), (U_{i_2}, V_{j_2}), ..., (U_{i_k}, V_{j_k}) >$ of pairs such that U_{i_x} and V_{j_x} are equivalent and for every two pairs (U_{i_x}, V_{j_x}) and (U_{i_y}, V_{j_y}), $i_x < i_y$ iff $j_x < j_y$. Define $\widetilde{U_{i_x}} = U_{i_x+1}...U_{i_{x+1}-1}$ (or ϵ if $i_{x+1} = i_x + 1$), and $\widetilde{U_0} = U_1...U_{i_1-1}$, and define $\widetilde{(V_{j_x})}$ similarly. Finally define $A_x = U_{i_x}$ and $A'_x = (\widetilde{(U_{i_x})}|\widetilde{(V_{j_x})})$. The* merge operator $\bigoplus$ *is defined to be the combination function that takes two regular expressions and returns a "covering" regular expression as follows: $U \bigoplus V = A'_0A_1A'_1...A_kA'_k$.*

For example $AB \bigoplus BC = A?BC?$ and $(ABCE) \bigoplus (B(C|D)E) = (A?B(C|D)E)$. Sequence alignment is a well-studied problem, and there are various solutions for it [11].

Definition 4. *Given a regular expression of the form $E = U^{[l_1,u_1]}CV^{[l_2,u_2]}$ where $(U \bigoplus V)C?$ is repeatable, the* compression function Γ *returns the regular expression $\Gamma(E) = ((U \bigoplus V)C?)^{[l_1+l_2,u_1+u_2]}$. A regular expression is* compressed *if no compression can be applied to any of its subexpressions, and in that case we define $\Gamma(E) = E$ for notational convenience. Note that $L(E) \subseteq L(\Gamma(E))$.*

For example, assume that $\omega(A) = \omega(B) = 3$, $\omega(C) = 1$, and $\alpha = 0.5$. $\Gamma(ABAB) = (AB)^2$, $\Gamma(ABC(AB)^3) = (ABC?)^4$, $\Gamma((ABC)^{[1,3]}(AB)^2) = (ABC?)^{[3,5]}$, and $\Gamma((AB)^+C(AB)^2) = ((ABC?)^{[3,\infty]}$. $ABABBCBC$ and $(AB)^2BCBC$ are not compressed, but $(AB)^2(BC)^2$ is compressed. If $\omega(A) = \omega(C) = 1$ and $\omega(B) = 3$, then $(AB)^2(BC)^2$ is not compressed because $AB \bigoplus BC$ is repeatable and $(A?BC?)^4$ is valid and compressed.

Definition 5. *For a constant k, a* k-normalized regular expression *(k-*NRE*) is a compressed regular expression where all bound repetition operators are of the form $[1, \infty]$ or $[m, n]$ where $m \leq n \leq k$. Regular tree grammars with k-normalized regular expressions are called* k-normalized regular tree grammars *(k-*NRTG*).*

For example $(AB)^3(CD)^2$ and $(AB)^{[1,3]}(CD)^2$ are compressed regular expressions, but they are not k-NREs for $k < 3$, $(AB)^+(CD)^2$ is a 2-NRE, and $(AB)^+(CD)^+$ is a 1-NRE. To ensure k-normality, we define the compression function, Γ_k, identically to Γ but replacing $[l, u]$ in the resulting expression by $[1, \infty]$ if $u > k$. Our experiments show that 2-NRTGs can capture the structure of most presentational XML published on the web.

We now briefly explain the concept of language identification in the limit introduced by Gold [9].

Definition 6. *A* language learner *is an algorithm that infers the name of an unknown language. The learner is provided with a training sequence $(i_1, i_2, \dots)$ of information units describing the language. A training sequence can be in the form of text (i.e., each i_t is a string from the language) or an informant (i.e., each i_t states whether or not a specific string is in the language). We associate a set of* allowable training sequences *with each language. Given a specific training sequence, at time t the learner is provided with i_t, and it guesses the name of the unknown language. The language is* identifiable in the limit *if after some finite time the guesses are all the same and are correct. A class of languages is* identifiable in the limit *if there is an effective learner (i.e. an algorithm) that can identify in the limit any language in the class given any allowable training sequence for that language.*

Note that in this definition a text consists only of positive examples while an informant can also contain negative examples. For example, consider these regular expressions: $R_1 = L((abc)^+)$ and $R_2 = L((abc?)^+)$. A text for the first regular expression can only contain strings such as *abc* or *abcabc*, whereas an informant can contain information such as "*abcabc* is in R_1" and "*ababc* is not in R_1." Since $R_1 \subset R_2$, if a learner is given only text from R_1, it cannot distinguish R_1 from R_2 (although, as we will show, the more samples from R_1 are observed with no samples from $R_2 - R_1$, the higher the chance that R_1 is the correct guess). Gold proves that the class of regular expression languages are not identifiable in the limit from positive examples. However, languages with limited cardinality are identifiable in the limit from positive examples. If L is identifiable in the limit, then a *characteristic set* of strings for L is a subset C of L such that for any training sample set S if $C \subseteq S$ then the learner identifies L in the limit.

The problem of inferring k-NRTGs from a small set of parse tree samples is equivalent to the problem of identifying an unknown language in the limit, but with the extra

requirement that the training set is finitely bounded and the bound is rather small. If the sample set does not contain a characteristic set of strings for the unknown language, then it is impossible to identify it, but we propose an approach that infers it with high probability.

3. THE INFERENCE ALGORITHM

3.1 Inferring k-NREs

The aim of this section is to infer a k-normalized regular expression (k-NRE) from a given set of sample strings over alphabet T. This is the basis for inferring a regular tree grammar for a given parse tree, as described in the next section.

We define an instance of a regular expression E to be an arbitrary string taken from $L(E)$. We represent non-terminals with capital letters and their instances with lower case letter, e.g. a is an instance of A.

THEOREM 1. *Consider a valid repetitive regular expression $R = E^+$ where $E = E_1E_2 \dots E_k$ and each E_i is an atomic regular expression that does not itself contain a repetitive pattern. An instance of R can be partitioned into instances of E in polynomial time. Moreover, E is identifiable in the limit, and if the set of its instances contains a characteristic set of E, R and E can be identified.*

PROOF. Note that any instance of R is a sequence of instances of E. Because none of the E_is contain a repetitive pattern, $L(E)$ has a limited cardinality (i.e. $|L(E)|$ is bounded), and so it can be finitely identified from positive examples. We show that the inference of E implies the inference of R.

Each E_i is an atomic regular expression that does not contain a repetitive regular expression. Thus, it is either a non-terminal, or it is disjunctive. Let $W_d = \sum \kappa(E_i)\omega(E_i)$ (the sum of the weights of the disjunctive components) and $W_n = \sum(1 - \kappa(E_i))\omega(E_i)$ (the sum of the weights of the non-terminals). R is a valid repetitive regular expression, so E is repeatable. Therefore $V(E) = \frac{W_d}{W_d+W_n} \leq \alpha$, so there are no disjunctive terms if $\alpha = 0$ and $W_d\frac{1-\alpha}{\alpha} \leq W_n$ if $0 < \alpha < 1$, which implies that $W_n > 0$. Thus some non-terminals must appear in every instance of E. Assume M is the set of such *mandatory* non-terminals, and let D be the set of non-terminals that appear in each E_i within R. $M \subseteq D$, so $0 < |M| \leq |D|$. Given R, we can find D in polynomial time (because of space limitation the algorithm is not presented here). After such *anchoring* non-terminals are recognized, some E_i that are in disjunctive form (or all such E_i if the set of E_i in R contains a characteristic set for E and thus $M = D$) can be determined. Consequently, E and R are recognized. □

In some cases where E contains boundary disjunctive forms (i.e. E_1 or E_k are disjunctive), ambiguities in recognizing instances of E might occur. For example $r = abababa$ can be partitioned as ab, ab, ab, a, which results in inferring $E = (AB?)^+$, or as a, ba, ba, ba, which results in $E = (B?A)^+$. Our algorithm generates all possible partitions, and heuristics can subsequently be applied to resolve such ambiguities.

For a given regular expression E with limited cardinality and sequence S of strings in $L(E)$, let $P(E|S)$ be the probability that E is the correct regular expression for describing S and $P(S|E)$ be the probability of generating S given E. Assuming all strings in $L(E)$ are generated with equal probability, $P(S|E) = \prod^{s\in S} P(s|E) = \prod^{|S|} \frac{1}{|L(E)|} = \frac{1}{|L(E)|^{|S|}}$. By Bayes' theorem, $P(E|S) = P(S|E)\frac{P(E)}{P(S)} = \frac{P(E)}{P(S)|L(E)|^{|S|}}$. If the repetitions of E within an instance of R do not contain a characteristic set for E, then E and hence R cannot be uniquely identified. Thus by following our probabilistic reasoning, the k-NRE with the smallest language that can generate S has the highest probability to be the unknown grammar. Hence, because R should be compressed, if $r = s_1s_2...s_n$ and if E_{s_i} is the regular expression chosen to describe s_i, $E_{s_1} \bigoplus E_{s_2} \cdots \bigoplus E_{s_n}$ has the highest probability of being the correct k-NRE for E. If $n > k$, then we choose $R = E^+$; otherwise, $R = E^n$ is more likely to be the correct expression for generating this instance.

Assume an instance of R occurs within a larger string, e.g., instance $abab$ of $(AB)^*$ in $cdababef$. Following a similar approach to the proof of Theorem 1, based on finding mandatory non-terminals in E, and relying on the fact that E is repeatable and the inferred grammar is compressed, we can find the instance of R within an arbitrary string, up to the ambiguity caused by boundary disjunctive forms.

Many ambiguities in finding an instance of R are caused by boundary disjunctive forms and can be resolved using a similar approach to the above, based on probabilities of generating strings. For example, assume a sample string is $s = abababc$. Two possibilities for E are $X_1 = (ABC?)$ and $X_2 = (AB)$. In the former case $abababc$ is an occurrence of R and in the latter case $ababab$ is an occurrence of R. Assume that R includes n repetitions of E (n is found as a consequence of finding the set of mandatory non-terminals). If the probability that c appears in an instance of X_1 is 0.5, the probability that it appears only in the last instance given n instances is $(0.5)^n$. Hence for that example $P((ab)^nc|X_1, n) = 0.5^n$ and $P((ab)^n|X_2, n) = 1$. Assuming no *a priori* bias (i.e., $P(X_1) = P(X_2)$), with a high probability X_2 represents E. Note, however, that if $\omega(C)$ is large enough, then there is no ambiguity because $(ABC?)$ is not repeatable and therefore is not an admissible solution. Alternatively, if $\omega(C)$ is small and c appears elsewhere within S, e.g. $s = abcababc$, then again there is no ambiguity: this time $X_1 = (ABC?)^+$ must be correct. Following a similar approach we can resolve most ambiguities caused by boundary disjunctive items in practice.

Given an input string S, if $\mathcal{R}$ is the set of so far identified repetitive patterns in S, for each $R_i \in \mathcal{R}$ we create a new temporary non-terminal N_i and replace the instance of R_i in S with n_i and repeat the above steps. The new non-terminals should be chosen in a way that allows merging compatible patterns if necessary. When no more repetitive patters are found, we replace the newly introduced non-terminals with their corrsponding regular expressions to generate the final result. This allows the inference of expressions with nested repetitive patterns such as $((AB)^+(CD?E)^+)^+$.

The above steps explain how a k-NRE can be inferred from a single input string. If we are given multiple input strings, we infer a regular expression for each and combine the results using $\bigoplus$.

3.2 Inferring k-NRTGs

Recall that each subtree in a presentational XML document is assumed to be an instance tree that is generated by a non-terminal in an unknown k-NRTG, and the sequence

Source	No. of samples	Our algorithm	MDR/ DEPTA	Modified DEPTA	NET
mls.ca	31	96%	26%	52%	77%
buzzle.com	62	91%	29%	59%	64%
imdb.com	60	96%	17%	55%	75%
movietimes.ca	9	89%	44%	56%	77%
amazon.com	105	99%	22%	52%	85%
dblp.com	15	94%	47%	73%	86%

Table 1: Comparison of various algorithms.

of children for its root is generated by the k-NRE for its content model. Therefore, to infer a grammar for a sample tree given as input, for each node we should infer a k-NRE for its sequence of children. Our algorithm traverses the tree in postorder so that productions for the children of a node are generated before the root node is considered.

Consider an instance tree T, a node N in T, and the sequence S of the subtrees rooted at the children of N, $S = c_1 \dots c_k$. Because of the postorder processing, k-NRTGs for each c_i have already been recognized. We use *hierarchical agglomerative clustering* [10], which is based on the similarity between trees, to partition the c_is into clusters such that each cluster contains similar subtrees that are assumed to be generated by a single production in the underlying tree regular grammar. (Thus all subtrees in a cluster must have the same root label.) There are various ways to calculate tree similarity, such as comparing the inferred k-NRTGs for the c_is using variations of *tree edit distance* [3]. Although tree edit distance usually provides a good measure of similarity, its calculation is costly (its complexity is $O(mn)$ where m and n are the sizes of the two trees). Alternatively, similarity can be defined based on heuristics such as comparing the label and degree of the roots, the sizes, and the depths of trees. Thus an appropriate similarity function should be chosen based on the application requirements.

We assign a unique id to each cluster, which allows us to map $c_1 \dots c_k$ to a sequence I of cluster ids. We combine the k-NREs for all trees in a cluster using the $\bigoplus$ operator. We assign a weight to each cluster id according to the resulting grammar as described in Section 2. Then, we use the algorithm from Section 3.1, to infer a regular expression for I. Lastly, for each cluster id we create a new non-terminal, with the cluster's common root label as its root label and the inferred k-NRE as its content model.

4. EXPERIMENTAL RESULTS

In this section we present the results of the empirical evaluation of our algorithm. We apply our algorithm to two types of presentational XML documents: one using XHTML and the other using Presentation MathML.

We compare our algorithm with state-of-the-art data extractors, namely DEPTA [16] and MDR [12], both of which apply very similar methods for detecting data records based on the DOM presentation of an HTML page, and with NET [13], which has also been proposed to detect and extract data records from web pages containing a list of at least two data records. Note that these are data extraction algorithms that consist of various modules for detecting data regions in documents, detecting data records, and so on. We restricted our implementation of the algorithms to those parts that perform grammar inference. As published, both MDR and DEPTA stop at each node when they recognize a repetitive pattern, and they do not continue to the children of that node. As a result, they do not recognize nested repetitive patterns, even if such patterns are simple. For comparison purposes, we modified them to continue to the children of a node when a repetitive pattern is found, and call it *modified DEPTA*.

We tried these algorithms on pages from various websites (listed in Table 1). From each website we downloaded some arbitrary pages, and converted each page into XHTML. From each web page we chose some prototypical sections (mostly with repeating patterns) and manually built a grammar for each section. The reason that we considered smaller granularity than a whole page to evaluate the algorithms is that XHTML pages are typically very large, and if an algorithm fails to correctly infer the correct structure of any part, we will conclude that the algorithm fails to infer the structure of the page as a whole. Moreover, in applications depending on data extraction, it often suffices if the algorithm can correctly infer structures for specific sections, namely the data-centric parts of a page. We chose mostly sections that contain repetitive patterns because such sections tend to have more complex structures and they form the basis for recognizing "data records." The success rate of an algorithm is the percentage of samples whose inferred grammars are equivalent to the corresponding manually created ones.

To begin, we investigate the effect of α (Definition 2) on the performance of our algorithm. Figure 3 shows how the performance of our algorithm changes as the value of α increases from 0 to 1 for two data sets of Presentation MathML and XHTML documents. As the figure implies, the performance for very small values of α is relatively poor because in these cases disjunctive forms are not allowed, or are very restricted, and hence the repetitive patterns that require disjunctive forms are not detected correctly. On the other hand, when α is very close to 1, disjunctive forms are less restricted, and some patterns that are not repetitive are mistakenly recognized as being repetitive patterns. Our algorithm performs best when α is between 0.2 and 0.5, and thus we set the value of α to 0.3 for our experiments.

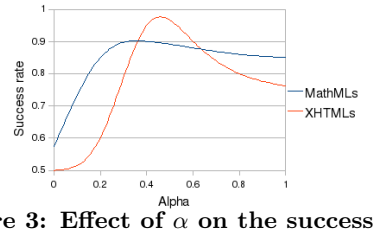

Figure 3: Effect of α on the success rate.

Some examples of regular expressions taken from various websites are listed in Table 2. For each algorithm, the table shows whether it can correctly recognize the regular expression. The number of repetitions for each pattern that is marked by an under-brace is also shown. The first two examples are quite complex, and they contain optionals, repetitive patterns, and duplicates. Our algorithm is the only one that can correctly recognize such patterns. In Row 6, $(AT?)^+$ is not valid for $\alpha = 0.3$, and hence our algorithm cannot detect it (although we note that for $\alpha \geq 0.5$, it does correctly recognize this pattern). Modified DEPTA (and hence MDR/DEPTA) cannot correctly recognize repetitive patterns that contain optional components. NET cannot infer the correct grammar in Rows 1, 3, and 7 because they contain repetitive patterns with duplicate non-terminals (in row 8, D appears twice in the content model of Y). Table 1 summarizes the performance of all the algorithms. (DEPTA and MDR use very similar algorithms for mining patterns; thus we represent the results for both in one column.)

Source	Regular expression	No. of repeats	Our algorithm	Modified DEPTA	NET
buzzle.com	E_1	5	√	×	×
	E_2	3	√	×	×
	where $E_1 = (T?A)^+T\underbrace{(B(RT)?RIUIRTR)^+}TAT$ and $E_2 = (T?A)^+TR\underbrace{(BR(TATR)?BOBUBNR)^+}TAT$				
imdb.com	$\underbrace{(ATI?)^+}$	26	√	×	√
	$\underbrace{(ATR)^+}$	2	√	√	√
	$\underbrace{(AU)^+}$	2	√	×	√
amazon.com	$\underbrace{H^+}$	2	× $\alpha = 0.3$		
			√ $\alpha \geq 0.5$	×	√
tribute.ca	$\underbrace{(YR?)^+}$	2	√	×	×

where

$A \rightarrow$ < a >$[T]$ $B \rightarrow$ < b >$[T]$ $D \rightarrow$ < div >$[T]$
$G \rightarrow$ < strong >$[T]$ $H \rightarrow$ < li >$[(AT?)^+]$ $I \rightarrow$ < i >$[T]$
$L \rightarrow$ < li >$[T]$ $M \rightarrow$ < li >$[T(AT)?]$ $N \rightarrow$ < ol >$[L^+]$
$O \rightarrow$ < ul >$[M^+]$ $P \rightarrow$ < span >$[G]$ $R \rightarrow$ < br >
$S \rightarrow$ < span >$[ATA]$ $T \rightarrow text$ $U \rightarrow$ < ul >$[L^+]$
$Y \rightarrow$ < div >$[SPT(DTDR?)^+]$

Table 2: Examples from various web pages.

Next, we apply our algorithm to mathematical expressions encoded with Presentation MathML.The first set contains 50 expressions that contain matrices, and the second set contains 50 expressions that contain polynomials or series. We normalized the expressions by removing values of numbers and names of variables, e.g. $x + 1$ is normalized to "$\{variable\}$ + $\{number\}$." Then, we applied our algorithm to each expression. Polynomials and series generally have more irregular structures, so our algorithm has a somewhat lower success rate for this set (see Table 3). In Table 4 some examples of mathematical expressions are shown. For each expression, a regular tree grammar that generates it is also shown (N and V in these examples are non-terminals that generate numbers and variables). Our algorithm does not detect a repetitive pattern in the last example, because the regular tree grammar that generates it is not valid for $\alpha = 0.3$.

Matrices	89 %	Polynomials and series	84%

Table 3: Success rate for mathematical expressions.

Expression	Regular expression	
$p_n = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!}$	$ILN(PF)^+$	√
$2x^2 + 3y^2 + 2z^2 = 1$	$(NJP?)^+LN$	√
$\begin{bmatrix} \lambda_1 & \gamma_1 & \beta_1 \\ \lambda_2 & \gamma_2 & \beta_2 \\ \lambda_3 & \gamma_3 & \beta_3 \end{bmatrix}$	Z^+	√
$\phi = t^2 - \frac{1}{c}\omega t^3 - t^4 - \frac{1}{c}\omega t^5$	$VL(U?G?V?E)^+$	×

where

$D \rightarrow <mtd> [I]$ $E \rightarrow <msup> [VN]$ $F \rightarrow <mfrac> [NM]$
$G \rightarrow <mfrac> [NV]$ $I \rightarrow <msub> [VN]$ $J \rightarrow <msup> [IN]$
$L \rightarrow <mo> =$ $M \rightarrow <mrow> [NO]$ $O \rightarrow <mo> !$
$P \rightarrow <mo> +$ $U \rightarrow <mo> -$ $Z \rightarrow <mtr> [D^+]$
$+ \rightarrow$ " + " $= \rightarrow$ " = " $! \rightarrow$ "!"

Table 4: Grammars for mathematical expressions.

5. CONCLUSIONS AND FUTURE WORK

We have defined a systematic approach to inferring grammars for presentational XML documents with complex tree structures such as XHTML Web pages and Presentation MathML expressions. We have shown that our algorithm outperforms three state-of-the-art data extraction tools, and also performs well on mathematical expressions.

We proposed solutions to resolve some ambiguities that arise when repeating patterns contain boundary disjunctive forms. We continue to search for and investigate examples of similar ambiguities in practice and to develop approaches to resolve them. Some presentation XML documents contain arbitrary and usually unimportant parts, e.g. advertisements, that have possibly different layouts and contents in each occurrence. We wish to modify our algorithm to detect such parts as special wild-card non-terminals.

6. REFERENCES

[1] G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. *ACM Trans. Web*, 4(4):14:1–14:32, 2010.

[2] G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of concise DTDs from XML data. In *VLDB*, pages 115–126, 2006.

[3] P. Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, 2005.

[4] C.-H. Chang and S.-C. Lui. IEPAD: Information extraction based on pattern discovery. In *WWW '01*, pages 681–688.

[5] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. 2007.

[6] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *VLDB '01*, pages 109–118.

[7] X. Gao, L. P. B. Vuong, and M. Zhang. Automatic data record detection in web pages. In *KSEM'07*, pages 349–361, Berlin, Heidelberg, 2007. Springer-Verlag.

[8] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: Learning document type descriptors from XML document collections. *Data Min. Knowl. Discov.*, 7(1):23–56, 2003.

[9] E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[10] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[11] H. Li and N. Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, 11(5):473–483, 2010.

[12] B. Liu, R. Grossman, and Y. Zhai. Mining data records in web pages. In *KDD '03*, pages 601–606.

[13] B. Liu and Y. Zhai. NET - a system for extracting web data from flat and nested data records. In *WISE '05*, pages 487–495, 2005.

[14] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Technol.*, 5(4):660–704, 2005.

[15] J. Wang and F. H. Lochovsky. Data extraction and label assignment for web databases. In *WWW '03*, pages 187–196, 2003.

[16] Y. Zhai and B. Liu. Structured data extraction from the web based on partial tree alignment. *IEEE Transactions on Knowledge and Data Engineering*, 18:1614–1628, 2006.

[17] S. Zheng, R. Song, J.-R. Wen, and D. Wu. Joint optimization of wrapper generation and template detection. In *KDD '07*, pages 894–902.