Information Extraction from Web Documents Based on Local Unranked Tree Automaton Inference

Raymond Kosala[1], **Maurice Bruynooghe**[1], **Jan Van den Bussche**[2], **Hendrik Blockeel**[1]
[1] K.U.Leuven, Dept. of Computer Science, Celestijnenlaan 200A, B-3001 Leuven
{raymond,maurice,hendrik}@cs.kuleuven.ac.be
[2] University of Limburg, Dept. WNI, Universitaire Campus, B-3590 Diepenbeek
jan.vandenbussche@luc.ac.be

Abstract

Information extraction (IE) aims at extracting specific information from a collection of documents. A lot of previous work on IE from semi-structured documents (in XML or HTML) uses learning techniques based on strings. Some recent work converts the document to a ranked tree and uses tree automaton induction. This paper introduces an algorithm that uses unranked trees to induce an automaton. Experiments show that this gives the best results obtained so far for IE from semi-structured documents based on learning.

1 Introduction

Information extraction aims at extracting specific information from a collection of documents. One can distinguish between IE from unstructured and from (semi-) structured texts [Muslea, 1999]. Extracting information from web documents belongs to the latter category and gains importance [Levy *et al.*, 1998]. These documents are not written in natural language, but rather involve explicit annotations such as HTML/XML tags to convey the structure of the information, making the methods tuned towards natural language unusable.

While special query languages exist [Bry and Schaffert, 2002; XQL, 2002], their use is time consuming and requires nontrivial skill. As argued in [Muslea *et al.*, 2001; Kushmerick, 2000], there is a need for systems that learn to extract information from a few annotated examples (*wrapper induction*). Several machine learning techniques for inducing wrappers have been proposed. Examples are multistrategy approaches [Freitag, 2000] and various grammatical inference techniques that induce a kind of delimiter-based patterns [Muslea *et al.*, 2001; Freitag and McCallum, 1999; Freitag and Kushmerick, 2000; Soderland, 1999; Freitag, 1997; Hsu and Dung, 1998; Chidlovskii *et al.*, 2000]. All these methods treat the document as a string of characters.

Structured documents such as HTML and XML documents, however, have an explicit tree structure. In [Kosala *et al.*, 2002b; 2002a], it is argued that one can better exploit this tree structure and use tree automata [Comon *et al.*, 1999]. The document tree is converted in a ranked binary tree and k-testable tree automata [Rico-Juan *et al.*, 2000] are induced and then used for the extraction task. Typically in a IE task from structured documents, there is some structural context close to the target. After linearisation in a string, this context can be arbitrarily far away, making the learning task very difficult for string based methods. While binarisation may also increase the distance between the context and the target, they remain closer, and the learning task should be easier. This is confirmed by the experiments in [Kosala *et al.*, 2002b; 2002a]. If distance between the relevant context and the target is indeed a main factor determining the ability to learn an appropriate automaton, then an algorithm inducing a wrapper directly from the unranked tree should perform even better. This path is pursued in the current paper. As in [Kosala *et al.*, 2002b; 2002a] user intervention is limited to annotating the field to be extracted in a few representative examples. String based methods require substantially more user intervention, such as splitting the document into small fragments, and selecting some of them for use as a training example, *e.g.* [Soderland, 1999]; the manual specification of the length of a window for the prefix, suffix and target fragments [Freitag and McCallum, 1999; Freitag and Kushmerick, 2000], and of the special tokens or landmarks such as "$>$" or ";" [Freitag and Kushmerick, 2000; Muslea *et al.*, 2001].

The rest of the paper is organized as follows. Section 2 provides some background on unranked tree automata and their use for IE. Section 3 describes our methodology and introduces our unranked tree inference algorithm. Results are described in Section 4, related work in Section 5. Section 6 concludes.

2 Preliminaries

Grammatical inference and information extraction. Grammatical inference (also called automata induction, grammar induction, or automatic language acquisition) refers to the process of learning rules from a set of labeled examples. The target domain is a formal language (a set of strings over some alphabet Σ) and the hypothesis space is a family of grammars. The inference process aims at finding a minimum automaton (the *canonical automaton*) that is *compatible* with the examples. There is a large body of work on grammar induction, for a survey see *e.g.* [Murphy, 1996; Parekh and Honavar, 1998].

In grammar induction, we have a finite alphabet Σ and a formal language $L \subseteq \Sigma^*$. Given a set of examples in L (S^+)

and a (possibly empty) set of examples not in L (S^-), the task is to infer a deterministic finite automaton (DFA) that accepts the examples in S^+ and rejects those in S^-. In [Freitag, 1997], an IE task is mapped into a grammar induction task. A document is converted into a sequence of tokens (from Σ). Examples are transformed by replacing the token to be extracted by the special token x. Then a DFA is inferred that accepts all the transformed examples. In our case, a document is converted into a tree and the token to be extracted (at a leaf) is replaced by the special token x. Then a tree automaton is inferred that accepts all the transformed examples. When using the learned automaton, a similar transformation is done. Each token that is a candidate for extraction is in turn replaced by x. The token replaced by x is extracted iff the transformed document is accepted by the automaton.

Unranked tree automata. Some existing algorithms for string automaton induction have been upgraded to ranked tree automaton induction (*e.g.* [Rico-Juan *et al.*, 2000; Abe and Mamitsuka, 1997])[1]. By converting documents (which are unranked) into binary trees (which are ranked), tree automaton induction can be used for IE as shown in [Kosala *et al.*, 2002b]. The present work avoids binarisation and uses unranked trees. Unranked tree automata have been studied since the late 60's, see *e.g.* [Brüggemann-Klein *et al.*, 2001] for a survey. To our knowledge, algorithms for inducing them do not yet exist. This paper is a first step in this direction.

An unranked label is a label with a variable rank (arity). Thus the number of children is not fixed by the label. Given a set V of labels in an unranked alphabet, we can define V^T, the set of all (unranked) trees, as follows:

- a is a tree where $a \in V$.
- $a(u_1, \ldots, u_n)$ is a tree, where $a \in V$ and each u_i is a tree.

An unranked tree automaton (UTA) is a quadruple (V, Q, Δ, FS), where V is a set of unranked labels, Q is a finite set of states, $FS \subseteq Q$ is a set of final (accepting) states, and Δ is a set of transitions where each transition is of the form $v(e) \to q$, where $v \in V$, $q \in Q$, and e is a regular expression over Q.

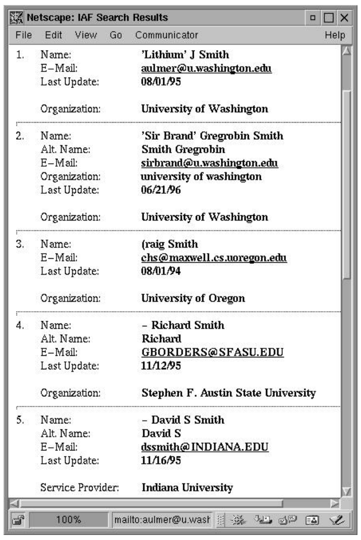
A bottom up UTA processes trees bottom up. When a leaf node is labeled v and there is a transition $v(e) \to q$ such that e matches the empty string, then the node is assigned state q. When an internal node is labeled v, its children have been assigned states $q_1, \ldots, q_n$, and there is a transition $v(e) \to q$ such that the string $q_1, \ldots, q_n$ matches the regular expression e, then the node is assigned state q. A tree is accepted if the state of its root is assigned an accepting state $q \in FS$.

3 Approach and algorithm

Preprocessing. Fig. 1 shows a representative task. For dealing with text nodes, we follow the approach described in [Kosala *et al.*, 2002b]: we replace most text nodes by `CDATA`[2], making an exception for so-called *distinguishing context*, specific text that is useful for the identification of the field of interest. *E.g.*, the text "Organization:" may be relevant for the extraction, hence this text should not be changed into CDATA. In each data set, at most one text field is identified as distinguishing context. It is found automatically using the approach described in [Kosala *et al.*, 2002b].

[1]Ranked: the label determines the number of children.

[2]See Fig. 2 for an example.


Netscape: IAF Search Results
File Edit View Go Communicator Help
100%
mailto:aulmer@u.wash

1. Name: 'Lithium' J Smith
E–Mail: aulmer@u.washington.edu
Last Update: 08/01/95

Organization: University of Washington

2. Name: 'Sir Brand' Gregrobin Smith
Alt. Name: Smith Gregrobin
E–Mail: sirbrand@u.washington.edu
Organization: university of washington
Last Update: 06/21/96

Organization: University of Washington

3. Name: (raig Smith
E–Mail: chs@maxwell.cs.uoregon.edu
Last Update: 08/01/94

Organization: University of Oregon

4. Name: – Richard Smith
Alt. Name: Richard
E–Mail: GBORDERS@SFASU.EDU
Last Update: 11/12/95

Organization: Stephen F. Austin State University

5. Name: – David S Smith
Alt. Name: David S
E–Mail: dssmith@INDIANA.EDU
Last Update: 11/16/95

Service Provider: Indiana University

Figure 1: The fields to be extracted are the fields following the `Alt.Name` and `Organization` fields. A document consists of a variable number of records. The number of occurrences of the fields to be extracted is variable (from zero to many). Also the position is not fixed.

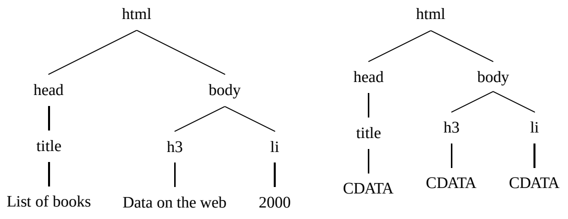


Figure 2: The left figure is an HTML tree. The right one is the same tree after abstracting the text nodes

Approach. Our approach for information extraction has the following characteristics:

- Strings stored at the text nodes are treated as single labels. If extracted, the whole string is returned.
- One automaton is learned for one type of field to be extracted, *e.g.*, the field following "Organization".
- In examples used during learning, one target field (a text node) is replaced by x. A document gives rise to several examples when several targets occur.

The learning phase proceeds as follows:

- Replace in the examples the target by "x", the distinguishing context(s) (if present) by "ctx" and all other text fields by `CDATA`.
- Map examples to trees and learn a tree automaton.

The extraction phase repeats for all candidate targets:

- Map the document to a tree and replace the candidate target by "x", the distinguishing context(s) (if present) by "ctx" and all other text fields by CDATA.

- Run the tree automaton; if the tree is accepted, then output the original text of the node labeled with x.

3.1 Local unranked tree automaton inference algorithm

Preliminaries. A *partition* of a set S is a set of disjoint nonempty subsets of S such that the union of the subsets is S. The height of a tree is defined as: $height(v) = 1$ and $height(v(t_1, \ldots, t_n)) = 1 + \max(height(t_1), \ldots, height(t_n))$. If $t = l(t_1, \ldots, t_n)$ then the t_i are called subtrees of t, and subtrees of t_i are also subtrees of t. When a tree t is "cut off" at level k, this means all subtrees at level k become leaves. Thus the height of the cut-off tree can be at most k. Given a tree t, the set of *roots* $r_k(t)$ is the singleton containing t cut off at k; the set of *forks* $f_k(t)$ contains all subtrees of t of height at least k, cut off at k; and the set of *subtrees* $s_k(t)$ contains all subtrees of t of height at most k. Roughly, these sets respectively collect all subtrees of height k at the top, in the middle, and at the bottom of t. See [Rico-Juan *et al.*, 2000] for a formal definition.

Example 1 *Let* $t = a(b(a(b, x)), c)$ *then* $r_1(t) = \{a\}$, $f_2(t) = \{a(b, c), b(a), a(b, x)\}$, *and* $s_1(t) = \{b, x, c\}$.

Algorithm 1 Local unranked tree inference algorithm

```
Input: Positive examples (set $T$) and an integer $k_c > 1$.
Output: An unranked tree automaton $(V, Q, \Delta, FS)$.
 1: $FS = \mathcal{F} = \mathcal{S} = \emptyset$;
 2: for each $t \in T$ do
 3:     $t' = convert\_labels(t)$
 4:     $\mathcal{F} = \mathcal{F} \cup f_2(t')$  % 2-forks
 5:     $\mathcal{S} = \mathcal{S} \cup s_1(t')$  % 1-subtrees
 6:     $FS = FS \cup r_1(t')$  % 1-root
 7: end for
 8: $Q = FS \cup \{r_1(f) | f \in \mathcal{F}\} \cup \mathcal{S}$
 9: $\Delta = \{v(\epsilon) \to v | v \in \mathcal{S}\}$
10: $P = partition(\mathcal{F})$
11: for each $(v, Str) \in P$ do
12:     if $v = v_x$ and the length of the shortest string in $Str >= k_c$
        then
13:         $e = k\_contextual(k_c, Str)$     % a DFA
14:     else if $v = v_x$ then
15:         $e = k\_contextual(2, Str)$     % a DFA
16:     else
17:         $e = *$       % accepts any sequence
18:     end if
19:     $\Delta = \Delta \cup \{v(e) \to v\}$
20: end for
```

The algorithm. We have designed a procedure, which is shown in Algorithm 1, to learn an unranked tree automaton from a set T of positive examples. The inferred automaton is "local" in the sense that it identifies a tree with its 2-forks as defined above. Note that DTDs for XML do the same [Murata *et al.*, 2001]. In addition to the input T, it takes as additional parameter a positive integer $k_c > 1$, which is the parameter for the k_contextual subroutine that it calls.

In a first for-loop, our algorithm collects all 2-forks, 1-subtrees and 1-roots. The latter become final states. However, before these steps, the node labels of each input tree t are rewritten using the function $convert\_labels(t)$. This function (Algorithm 2) rewrites node labels. The label v of the root of a subtree is changed into v_x if that subtree contains a "x" and into v_{ctx} if that subtree contains a "ctx". The effect is that the special labels are remembered up to the root[3].

Algorithm 2 convert_labels(t)

```
Input: A tree $t = v(t_1, \ldots, t_n)$.
Output: A tree $t'$
 1: $t'_1 = convert\_labels(t_1), \ldots, t'_n = convert\_labels(t_n)$
 2: if $\exists i : t'_i$ has label $l_x$ or $x$ then
 3:     $t' = v_x(t'_1, \ldots, t'_n)$
 4: else if $\exists i : t'_i$ has label $l_{ctx}$ or $ctx$ then
 5:     $t' = v_{ctx}(t'_1, \ldots, t'_n)$
 6: else
 7:     $t' = t$
 8: end if
```

Algorithm 3 k_contextual(k, Str)

```
Input: An integer $k > 1$ and set of strings $Str$ over $\Sigma$
Output: A $k$-contextual DFA $(\Sigma, Q, \Delta, FS)$.
 1: $Q = FS = \Delta = \emptyset$;
 2: $U = \{\#^{k-1}s\# | s \in Str, \# \notin \Sigma\}$
 3: $Grams = \{s | s$ is a substring of $u, u \in U, |s| = k\}$
 4: for each $g \in Grams$ do
 5:     Let $g = uv$ with $|u| = 1$ and $|v| = k - 1$
 6:     $Q = Q \cup \{v\}$
 7:     if $v = v'\#$ then
 8:         $FS = FS \cup \{v\}$
 9:     end if
10:     $\Delta = \Delta \cup \{uv \to v\}$
11: end for
```

Next, the states are collected (the 1-root, the 1-subtrees, and the 1-roots of the forks) in Q, the transitions are initialized with one transition for each 1-subtree and the 2-forks are partitioned according to the label of the forks' root. The latter results in a set of pairs (v, Str) with Str a set of sequences, each sequence being the children of a fork. *E.g.*, $\{a, \{(b, c), (b, c, c)\}\}$ represents two forks with root label a.

In the second for-loop, the k_contextual algorithm (Algorithm 3) [Muggleton, 1990; Ahonen, 1996] is used to learn a deterministic finite automaton (DFA) that can be used as the representation of the regular expressions e to be used in the transitions $v(e) \to q$ of the UTA. As an illustration, consider an input string ab for $k = 3$. The value of U is $\#\#ab\#$ and one obtains $FS = \{b\#\}$ and $\Delta = \{\#\#a \to \#a, \#ab \to ab, ab\# \to b\#\}$, where $\#$ is a distinguished label that is not in Σ. It is capable of identifying in the limit any k-contextual string automaton, a subset of the finite automata, from positive examples only. In principle, we could use any string automaton inference algorithm for this purpose (see *e.g.* [Murphy, 1996; Parekh and Honavar, 1998] for others). We choose the k_contextual algorithm because it is efficient, simple, and works well in practice.

[3]As a result, the tree automaton is not purely local.

An element (v, Str) of the partition gives all positive examples for a particular label v. The regular expression e captures the regularity in these examples (the document content model in XML terminology). To obtain sufficient generalization, we decided, after some experimentation, to distinguish three cases. If all children of a v_x node are long enough, we construct a DFA using the k-contextual algorithm with k-value k_c, otherwise with value 2. For other labels (either a v_{ctx} label or an original label), we ignore the content of the children and accept any sequence (the regular expression "$*$").

Using a result of [Muggleton, 1990], it is quite straightforward to make an incremental version of Algorithm 1. We omit it here due to lack of space. Using a basic result of [Angluin, 1980], we can prove:

Theorem 1 *Every unranked tree language that is definable by a local unranked tree automaton with k-contextual regular expressions is identifiable in the limit, from positive examples only, by our algorithm.*

Example 2 *Applying the Algorithm 1 on the tree of Example 1 for $k_c = 2$, we obtain:*

- $t' = a_x(b_x(a_x(b, x)), c)$
- $r_1(t') = \{a_x\}$
- $\mathcal{F} = f_2(t') = \{a_x(b_x, c),\ b_x(a_x),\ a_x(b, x)\}$
- $\mathcal{S} = s_1(t') = \{b,\ x,\ c\}$.
- $FS = \{a_x\}$
- $Q = \{a_x,\ b_x,\ b,\ x,\ c\}$
- $P1 = \{a_x, \{(b_x, c), (b, x)\}\}$
- $P2 = \{b_x, \{(a_x)\}\}$
- $DFA1 : \Delta = \{\#, b_x \rightarrow b_x;\ b_x, c \rightarrow c;\ c, \# \rightarrow \#;\ \#, b \rightarrow b;\ b, x \rightarrow x;\ x, \# \rightarrow \#\}$, $Q = \{b_x, c, \#, b, x\}$, $FS = \{\#\}$.
- $DFA2 : \Delta = \{\#, a_x \rightarrow a_x;\ a_x, \# \rightarrow \#\}$, $Q = \{a_x, \#\}$, $FS = \{\#\}$.
- *transitions:*
  - $b \in \mathcal{S} : b(\epsilon) \rightarrow b$
  - $x \in \mathcal{S} : x(\epsilon) \rightarrow x$
  - $c \in \mathcal{S} : c(\epsilon) \rightarrow c$
  - $P1 : a_x(DFA1) \rightarrow a_x$
  - $P2 : b_x(DFA2) \rightarrow b_x$

4 Experimental results

We evaluated our method on two semi-structured data sets commonly used in IE research (available online from http://www.isi.edu/~muslea/RISE/): a collection of web pages containing people's contact addresses (the Internet Address Finder (IAF) database) and a collection of web pages about stock quotes (the Quote Server (QS) database). For each dataset, there are two tasks; they are the extraction of alternative and organization fields in the IAF dataset and of the date and volume fields in the QS dataset. Each dataset consists of 10 documents. The number of fields to be extracted is respectively 94 (IAF-organization), 12 (IAF-alt.name), 24 (QS-date), and 25 (QS-vol). We choose these datasets because they are benchmark datasets that are commonly used for research in IE; hence they allow us to compare results. In order to provide a close comparison, we use the same train and test splits as in [Freitag and Kushmerick, 2000]. In addition, they require the extraction of a whole leaf node (our algorithms are designed for that task). Moreover, the results obtained so far [Muslea *et al.*, 1999; Hsu and Chang, 1999] indicate that they are difficult tasks. In fact one of the authors in [Muslea *et al.*, 1999] has tried to build a hand-crafted extractor given all available documents from the QS dataset and achieved only 88% accuracy (or recall in our criteria below). We also test our method on a significantly reduced Shakespeare XML dataset (available online from http://www.ibiblio.org/bosak/). We use the same training and test set as in [Kosala *et al.*, 2002b]. The task on this dataset is to extract the second scene from one act in a particular play.

We apply the commonly used criteria of IE research for evaluating our method. Precision P is the number of correctly extracted objects divided by the total number of extractions, while recall R is the number of correct extractions divided by the total number of objects present in the answer template. The F1 score is defined as $2PR/(P + R)$, the harmonic mean of P and R. Table 1 shows the results we obtained as well as those obtained by some current state-of-the-art string-based methods: an algorithm based on Hidden Markov Models (HMMs) [Freitag and McCallum, 1999], the Stalker wrapper induction algorithm [Muslea *et al.*, 2001] and BWI [Freitag and Kushmerick, 2000]. We also include the results of the k-testable algorithm in [Kosala *et al.*, 2002b] which works on ranked trees. The results of HMM, Stalker and BWI are taken from [Freitag and Kushmerick, 2000]. All tests are performed with 10-fold cross validation following the splits used in [Freitag and Kushmerick, 2000], except in the small Shakespeare dataset which uses 2-fold cross validation. Each split has 5 documents for training and 5 for testing. We refer to Section 5 for a description of these methods.

Table 1 shows the best results of the unranked method with a certain k_c (cross-validation on one fold of 50% random training and test examples.) As can be seen, our method is the only one giving optimal results.

Table 2 shows the value of k and k_c used respectively by the k-testable and the unranked algorithm. It is well-known that when learning from positive examples only, there is a problem of over-generalization. Our algorithm requires a cross-validation on the value of k_c to avoid over-generalization.

Algorithm 1 runs in time O(cn), where n is the total number of nodes in the training examples and c is a constant. It takes an average time between 19 and 26 ms in a 1.7 Ghz Pentium 4 PC for Algorithm 1 to learn an example in the IAF and QS datasets.

5 Related work

The IE work for (semi-) structured texts can be divided into systems built manually using a knowledge engineering approach, *e.g.* [Hammer *et al.*, 1997] and systems built (semi-) automatically using machine learning techniques or other algorithms. The latter are called wrapper induction methods. We briefly survey them.

The three systems referred to in Table 1 learn wrappers based on regular expressions. BWI [Freitag and Kushmerick, 2000] uses a boosting approach in which the weak learner learns a simple regular expression with high precision but low

Table 1: Comparison of the results

	IAF - alt.name			*IAF - organization*			*QS - date*			*QS - volume*			*Small Shakespeare*		
	Prec	*Rec*	*F1*	*Prec*	*Rec*	*F1*	*Prec*	*Rec*	*F1*	*Prec*	*Rec*	*F1*	*Prec*	*Rec*	*F1*
HMM	1.7	90	3.4	16.8	89.7	28.4	36.3	100	53.3	18.4	96.2	30.9	-	-	-
Stalker	100	-	-	48.0	-	-	0	-	-	0	-	-	-	-	-
BWI	90.9	43.5	58.8	77.5	45.9	57.7	100	100	100	100	61.9	76.5	-	-	-
k-testable	100	73.9	85	100	57.9	73.3	100	60.5	75.4	100	73.6	84.8	56.2	90	69.2
unranked	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100

Table 2: Parameters used for the experiments

	IAF - alt.name	*IAF - org.*	*QS - date*	*QS - volume*	*Small Shakespeare*
k-testable (k)	4	4	2	5	3
unranked (k_c)	2	2	2	7	3

recall. The HMM approach [Freitag and McCallum, 1999] learns a hidden Markov model; it solves the problem of estimating probabilities from sparse data by using a statistical technique called shrinkage. This model has been shown to achieve state-of-the-art performance on a range of IE tasks. Stalker [Muslea *et al.*, 2001] induces extraction rules that are expressed as simple landmark grammars, a class of finite automata; it performs hierarchical extraction guided by a manually built *embedded catalog* tree that describes the structure of the fields to be extracted.

Several techniques based on naive-Bayes, two regular language inference algorithms, and their combinations for IE from unstructured texts are described in [Freitag, 1997]. WHISK [Soderland, 1999] learns extraction rules based on a form of regular expression patterns with a top-down rule induction technique. [Chidlovskii *et al.*, 2000] describe an incremental grammar induction approach; they use a subclass of deterministic finite automata that do not contain cyclic patterns. The SoftMealy system [Hsu and Dung, 1998] learns separators that identify the boundaries of the fields of interest. [Hsu and Chang, 1999] propose two classes of SoftMealy extractors: single pass, which is biased for tabular documents such as QS data (they reach up to 97% recall), and multi pass, which is biased for tagged-list document such as IAF data (they reach up to 57% recall). We cannot really compare results because the experimental setting is different.

All above methods use algorithms for learning string languages and require some manual intervention. HMMs and BWI require to specify a window length for the prefix, suffix and the target fragments. Stalker and BWI require to specify special tokens or landmarks such as ">" or ";". SoftMealy extractors in [Hsu and Chang, 1999] requires to choose between single and multi pass bias.

In [Kosala *et al.*, 2002b], the document is converted in a ranked (binary) tree and an algorithm is used that induces a k-testable tree automaton. However, as binarisation increases the distance between target and distinguishing context, large k are needed and the resulting automaton is precise but does not generalize enough (Table 1). In [Kosala *et al.*, 2002a], the same authors generalize the obtained automaton by selectively introducing wild-card labels. This gives some modest improvement in recall but does not solve the problem. Our unranked tree automaton induction algorithm does.

The most apparent limitation of our method is that it can only output a whole text node. To overcome this, it could be extended with a second step where string based methods are used to extract part of the text node. For example, to extract the substring "the web" from the whole string "Data on the web" (Fig. 2). Another limitation is that our method only output a single field (slot) in one run.

Finally, a disadvantage that is not apparent from the results reported above, is that when the identification of target fields does not require dependencies between nodes in the tree but can rely on a local pattern (e.g., the field to be extracted is always surrounded by specific delimiters), our tree based method needs more examples to learn the same extraction rule as methods that automatically focus on local patterns. Intuitively, more variations in further-away nodes need to be observed before these variations are considered irrelevant. This is simply an instance of the well-known trade-off between the generality of a hypothesis space and the efficiency with which the correct hypothesis can be extracted from it.

Some other approaches that exploit the structure of the documents are: WL^2 [Cohen *et al.*, 2002], a logic-based wrapper learner that uses multiple (string, tree, visual, and geometric) representations of the HTML documents. In fact, WL^2 is able to extract all four tasks in the IAF and QS datasets with 100% recall; and wrappers [Sakamoto *et al.*, 2002] that identify a field with a path from root to leaf, imposing conditions on each node in the path.

6 Conclusion

We have presented an algorithm for the inference of a local unranked tree automaton with k-contextual regular expressions and have shown that it can be used for IE from structured documents. Our results confirm the claim of [Kosala *et al.*, 2002b] that utilizing the tree structure of the documents is worthwhile for structured IE tasks. Whereas the latter work transforms the positive examples into binary ranked trees, we use them directly as unranked trees. Our results are optimal for the previously considered benchmarks, substantially improving upon the published results of other string and tree based methods and are a strong indication that unranked tree automata are much better suited than ranked ones for structured IE tasks.

Possible future work includes experiments with larger and more difficult datasets, adapting tree automata for multi slot extraction in one run, and a more formal analysis of the algorithm.

Acknowledgements
We thank Nicholas Kushmerick for providing us with the datasets used for BWI experiments and anonymous reviewers for their very valuable comments. This work is supported by the FWO project query languages for data mining. Hendrik Blockeel is a post-doctoral fellow of the Fund for Scientific Research of Flanders.

References

[Abe and Mamitsuka, 1997] N. Abe and H. Mamitsuka. Predicting protein secondary structure using stochastic tree grammars. *Machine Learning*, 29:275–301, 1997.

[Ahonen, 1996] H. Ahonen. *Generating grammars for structured documents using grammatical inference methods*. PhD thesis, University of Helsinki, Department of Computer Science, 1996.

[Angluin, 1980] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

[Brüggemann-Klein *et al.*, 2001] A. Brüggemann-Klein, M. Murata, and D. Wood. newblock Regular tree and regular hedge languages over unranked alphabets, TR 2001-05, HKUST-TCSC, 2001.

[Bry and Schaffert, 2002] F. Bry and S. Schaffert. Towards a declarative query and transformation language for XML and semistructured data: Simulation unification. In *ICLP 2002*, pages 255–270, 2002.

[Chidlovskii *et al.*, 2000] B. Chidlovskii, J. Ragetli, and M. de Rijke. Wrapper generation via grammar induction. In *ECML 2000*, pages 96–108, 2000.

[Cohen *et al.*, 2002] W. Cohen, M. Hurst, and L. S. Jensen. A flexible learning system for wrapping tables and lists in HTML documents. In *The Eleventh International World Wide Web Conference (WWW2002)*, 2002.

[Comon *et al.*, 1999] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. Available on: http://www.grappa.univ-lille3.fr/tata, 1999.

[Freitag and Kushmerick, 2000] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *AAAI/IAAI 2000*, pages 577–583. AAAI Press, 2000.

[Freitag and McCallum, 1999] D. Freitag and A. McCallum. Information extraction with HMMs and shrinkage. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.

[Freitag, 1997] D. Freitag. Using grammatical inference to improve precision in information extraction. In *ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition*, 1997.

[Freitag, 2000] D. Freitag. Machine learning for information extraction in informal domains. *Machine Learning*, 39(2/3):169–202, 2000.

[Hammer *et al.*, 1997] J. Hammer, H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha. Extracting semistructured information from the Web. In *Workshop on Management of Semistructured Data*, pages 18–25, 1997.

[Hsu and Chang, 1999] C-N. Hsu and C-C. Chang. Finite-state transducers for semi-structured text mining. In *IJCAI-99 Workshop on Text Mining: Foundations, Techniques and Applications*, 1999.

[Hsu and Dung, 1998] 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.

[Kosala *et al.*, 2002a] R. Kosala, M. Bruynooghe, H. Blockeel, and J. Van den Bussche. Information extraction by means of a generalized k-testable tree automata inference algorithm. In *Information Integration and Web-based Applications & Services, IIWAS, Proc.*, pages 105–109, 2002.

[Kosala *et al.*, 2002b] R. Kosala, J. Van den Bussche, M. Bruynooghe, and H. Blockeel. Information extraction in structured documents using tree automata induction. In *PKDD 2002*, pages 299–310, 2002.

[Kushmerick, 2000] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118:15–68, 2000.

[Levy *et al.*, 1998] A. Levy, C. Knoblock, S. Minton, and W. Cohen. Trends and controversies: Information integration. *IEEE Intelligent Systems*, 13(5), 1998.

[Muggleton, 1990] S. Muggleton. *Inductive Acquisition of Expert Knowledge*. Addison-Wesley, 1990.

[Murata *et al.*, 2001] M. Murata, D. Lee, and M. Mani. Taxonomy of XML schema languages using formal language theory. In *Extreme Markup Languages*, 2001.

[Murphy, 1996] K. Murphy. Learning finite automata. TR 96-04-017, Santa Fe Institute, 1996.

[Muslea *et al.*, 1999] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *Int. Conf. on Autonomous Agents*, 1999.

[Muslea *et al.*, 2001] I. Muslea, S. Minton, and C. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 4:93–114, 2001.

[Muslea, 1999] I. Muslea. Extraction patterns for information extraction tasks: A survey. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.

[Parekh and Honavar, 1998] R. Parekh and V. Honavar. *Automata Induction, Grammar Inference, and Language Acquisition*. Handbook of Natural Language Processing. New York: Marcel Dekker, 1998.

[Rico-Juan *et al.*, 2000] J.R. Rico-Juan, J. Calera-Rubio, and R.C. Carrasco. Probabilistic k-testable tree-languages. In *Grammatical Inference: Algorithms and Applications ICGI 2000,*, pages 221–228, 2000.

[Sakamoto *et al.*, 2002] H. Sakamoto, H. Arimura, and S. Arikawa. Knowledge discovery from semistructured texts. In *Progress in Discovery Science - Final Report of the Japanese Discovery Science Project*, pages 586–599. Springer, 2002.

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

[XQL, 2002] Xquery 1.0: An xml query language. W3C Working Draft 16 August 2002. `www.w3.org/TR/xquery/`.