Information Extraction in Structured Documents using Tree Automata Induction

Raymond Kosala[1], Jan Van den Bussche[2],
Maurice Bruynooghe[1], and Hendrik Blockeel[1]

[1] Katholieke Universiteit Leuven, Department of Computer Science
Celestijnenlaan 200A, B-3001 Leuven, Belgium
{Raymondus.Kosala, Maurice.Bruynooghe,
Hendrik.Blockeel}@cs.kuleuven.ac.be
[2] University of Limburg (LUC), Department WNI,
Universitaire Campus, B-3590 Diepenbeek, Belgium
jan.vandenbussche@luc.ac.be

Abstract. Information extraction (IE) addresses the problem of extracting specific information from a collection of documents. Much of the previous work for IE from structured documents formatted in HTML or XML uses techniques for IE from strings, such as grammar and automata induction. However, such documents have a tree structure. Hence it is natural to investigate methods that are able to recognise and exploit this tree structure. We do this by exploring the use of tree automata for IE in structured documents. Experimental results on benchmark data sets show that our approach compares favorably with previous approaches.

1 Introduction

Information extraction (IE) is the problem of transforming a collection of documents into information that is more readily digested and analyzed [6]. There are basically two types of IE: IE from unstructured texts and IE from (semi-) structured texts [15]. Classical or traditional IE tasks from unstructured natural language texts typically use various forms of linguistic pre-processing. With the increasing popularity of the Web and the work on information integration from the database community, there is a need for structural IE systems that extract information from (semi-) structured documents. Building IE systems manually is not feasible and scalable for such a dynamic and diverse medium as the Web [16]. Another problem is the difficulty in porting IE systems to new applications and domains if it is to be done manually. To solve the above problems, several machine learning techniques have been proposed such as inductive logic programming, e.g. [8, 2], and induction of delimiter-based patterns [16, 22, 7, 14, 3]; methods that can be classified as grammatical inference techniques.

Some previous work on IE from structured documents [16, 14, 3] uses grammatical inference methods that infer regular languages. However structured documents such as HTML and XML documents (also annotated unstructured texts) have a tree structure. Therefore it is natural to explore the use of tree automata

for IE from structured documents. Indeed, tree automata are well-established and natural tools for processing trees [5]. An advantage of using the more expressive tree formalism is that the extracted field can depend on its structural context in a document, a context that is lost if the document is linearized into a string.

This paper reports on the use of k-testable tree languages, a kind of tree automata formalism, for the extraction of information from structured documents.

In Section 2 we recall how information extraction can be formulated as a grammatical inference problem and provide some background on tree automata and unranked tree languages. Then in Section 3 we describe our methodology and give the details of the k-testable tree algorithm we have used for our prototype implementation. Finally we present our experimental results in Section 4, discussion and related work in Section 5 and conclusions in Section 6.

2 Tree grammar inference and information extraction

2.1 Grammatical inference

Grammatical inference refers to the process of learning rules from a set of labelled examples. It belongs to a class of inductive inference problems [20] in which the target domain is a formal language (a set of strings over some alphabet Σ) and the hypothesis space is a family of grammars. It is also often referred to as automata induction, grammar induction, or automatic language acquisition. It is a well-established research field in AI that goes back to Gold's work [11]. The inference process aims at finding a minimum automaton (the *canonical automaton*) that is *compatible* with the examples. The compatibility with the examples depends on the applied quality criterion. Quality criteria that are generally used are exact learning in the limit of Gold [11], query learning of Angluin [1] and probably approximately correct (PAC) learning of Valiant [24]. There is a large body of work on grammatical inference, see e.g. [20].

In regular grammar inference, we have a finite alphabet Σ and a regular language $L \subseteq \Sigma^*$. Given a set of examples that are in the language (S^+) and a (possibly empty) set of examples not in the language (S^-), the task is to infer a deterministic finite automaton (DFA) A that accepts the examples in S^+ and rejects the examples in S^-.

Following Freitag [7], we recall how we can map an information extraction task to a grammar inference task. We preprocess each document into a sequence of tokens (from an alphabet Σ). In the examples, the field to be extracted is replaced by the special token x. Then the learner has to infer a DFA for a language $L \subseteq (\Sigma \cup \{x\})^*$ that accepts the examples where the field to be extracted is replaced by x.

2.2 Tree languages and information extraction

Given a set V of ranked labels (a finite set of function symbols with arities), one can define a set of trees, denoted as V^T, as follows: a label of rank 0 is a tree,

if f/n is a label of rank $n > 0$ and $t_1, \ldots, t_n$ are trees, then $f(t_1, \ldots, t_n)$ is a tree. We represent trees by ground terms, for example with $a/2, b/1, c/0 \in V$, the tree below is represented by the term $a(b(a(c,c)),c)$.

A deterministic tree automaton (DTA) M is a quadruple (V, Q, Δ, F), where V is a set of ranked labels (a finite set of function symbols with arities), Q is a finite set of states, $F \subseteq Q$ is a set of final or accepting states, and $\Delta : \bigcup_k V_k \times Q^k \rightarrow Q$ is the transition function, where V_k denotes the subset of V consisting of the arity-k labels. For example, $\delta_k(v, q_1, \ldots, q_k) \rightarrow q$, where $v/k \in V_k$ and $q, q_i \in Q$, represents a transition.

A DTA usually processes trees bottom up. Given a leaf labeled $v/0$ and a transition $\delta_0(v) \rightarrow q$, the state q is assigned to it. Given a node labeled v/k with children in state $q_1, \ldots, q_k$ and a transition $\delta_k(v, q_1, \ldots, q_k) \rightarrow q$, the state q is assigned to it. We say a tree is accepted when a tree has at least one node with an accepting state $q \in F$ assigned to it.

Grammatical inference can be generalized from string languages to tree languages. Rather than a set of strings over an alphabet Σ given as example, we are now given a set of trees over a ranked alphabet V. Rather than inferring a standard finite automaton compatible with the string examples, we now want to infer a compatible tree automaton. Various algorithms for this kind of tree automata induction have been developed (e.g., [18, 19]).

A problem, however, in directly applying tree automata to tree-structured documents such as HTML or XML documents, is that the latter trees are "unranked": the number of children of a node is not fixed by the label, but is varying. There are two approaches to deal with this situation:

1. The first approach is to use a generalized notion of tree automata towards unranked tree formalisms (e.g., [17, 23]). In such formalisms, the transition rules are of the form $\delta(v, e) \rightarrow q$, where e is a regular expression over Q that describes a sequence of states.
2. The second approach is to encode unranked trees into ranked trees, specifically, binary trees, and to use existing tree automata inference algorithms for inducing the tree automaton.

In this paper we follow the second approach, because it seems less complicated. An advantage is that we can use existing learning methods that work on ranked trees. A disadvantage is that we have to preprocess the trees before applying the algorithm.

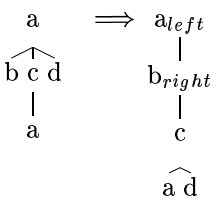
Using the symbol T to denote unranked trees and F to denote a sequence of unranked trees (a forest), the following grammar defines unranked trees:

$$T ::= a(F),\ a \in V \qquad\qquad F ::= \epsilon$$
$$F ::= T, F$$

There are well-known methods of encoding unranked trees by binary trees which preserve the expressiveness of the original unranked trees. The one we use can be formally defined with the following recursive function *encode* (with $encode_f$ for the encoding of forests):

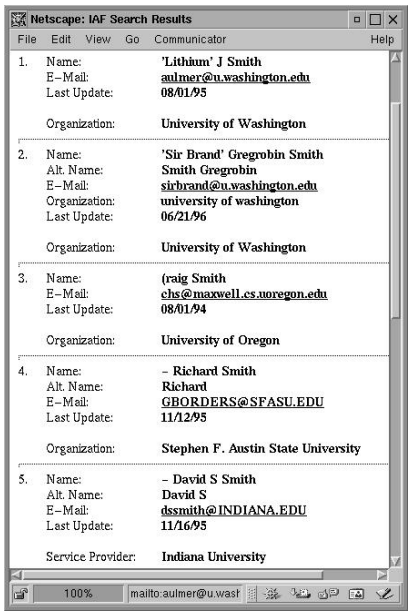
$$encode(T) \stackrel{\text{def}}{=} encode_f(T)$$

$$encode_f(a(F_1), F_2) \stackrel{\text{def}}{=} \begin{cases} a & \text{if } F_1 = F_2 = \epsilon \\ a_{right}(encode_f(F_2)) & \text{if } F_1 = \epsilon, F_2 \neq \epsilon \\ a_{left}(encode_f(F_1)) & \text{if } F_1 \neq \epsilon, F_2 = \epsilon \\ a(encode_f(F_1), encode_f(F_2)) & \text{otherwise} \end{cases}$$



Informally, the first child of a node v in an unranked tree T is encoded as the left child of the corresponding node v' of T', the binary encoding of T, while the right sibling of a node v in tree T is encoded as the right child of v' in T'. To distinguish between a node with one left child and a node with one right child, the node is annotated with left and right respectively. For example, the unranked tree $a(b, c(a), d)$ is encoded into binary tree $a_{left}(b_{right}(c(a, d)))$, or pictorially shown on the left. Note that the binary tree has exactly the same number of nodes as the original tree.

3 Approach and the algorithm



On the left, a simplified view of a representative document from the datasets that we use for the experiment is shown.[3] In this dataset, the fields to be extracted are the fields following the "Alt. Name" and "Organization" fields. A document consists of a variable number of records. In each record the number of occurrences of the fields to be extracted is also variable (from zero to several occurrences). Also the position where they occur is not fixed. The fact that the fields to be extracted follow the "Alt. Name" and "Organization" field suggest that the task is not too difficult. However this turns out not to be the case as we can see from the results of several state-of-the-art systems in Section 4.

Our approach for information extraction has the following characteristics:

- Some IE systems preprocess documents to split them up in small fragments. This is not needed here as the tree structure takes care of this: each node has an annotated string. Furthermore the entire document tree can be used as training example. This is different from some IE systems that only use a part of the document as training example.

[3] It is important to keep in mind that the figure only shows a rendering of the document, and that in reality it is a tree-structured HTML document.

- Strings stored at the nodes are treated as single labels. If extracted, the whole string is returned.
- A tree automaton can extract only one type of field, e.g. the field following 'Organization'. In order to extract multiple fields, a different automaton has to be learned for each field of interest.
- Examples used during learning contain a single node labeled with x. If the document contains several fields of interest, then several examples are created from it. In each example, one field of interest is replaced by an x.

The learning procedure is as follows:

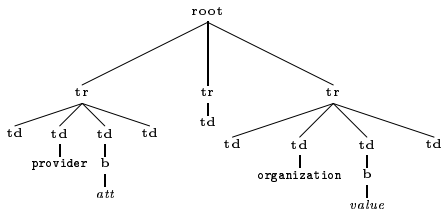
1. Annotate each example:
   - Replace the label of the node to be extracted by the special symbol x.
   - Parse the example document into a tree.
2. Run a tree automaton inference algorithm on the examples and return the inferred automaton.

The extraction procedure is as follows:

1. Parse the document into a tree.
2. Repeat for all text nodes:
   - Replace the text label of one text node by the special label x.
   - Run the automaton.
   - If the parse tree is accepted by the automaton, then output the original text of the node labeled with x.

Note that the extraction procedure can output several text nodes.

An implementation issue is how we deal with the contents of the various text nodes in the documents. The input to the algorithm consists of trees with *all* text string at the leaf changed to 'CDATA'[4] except one that we call *distinguishing context*. The abstraction of the text strings to CDATA is done to get a generalization of the tree patterns of the information that we want to extract. This could be easily done when parsing. Representing each different text string as a separate label is undesirable since it would lead to over-specification. Roughly speaking a distinguishing context is the text content of a node that is 'useful' for the identification of the field of interest. An example of the usefulness of the distinguishing context can be seen in the following depiction of a document of the kind already shown at the beginning of this Section:



[4] CDATA is the keyword used in XML document type descriptions to indicate text strings [25].

Suppose we like to extract the field 'value' and the text label `organization` always preceeds the field 'value'. In such case we call the text label `organization` a *distinguishing context* (for the field 'value'). If the labels `provider` and `organization` are both replaced by `CDATA` then any automaton that extracts the 'value' node will also extract the 'att' node. Indeed, the distinguishing context `provider` vs. `organization` has disappeared.

In our experiments we always use one distinguishing context for each field of interest when learning and testing the automaton. This is also the setting that we use for our experiments. The distinguishing context is chosen automatically. Our method is to find the invariant text label that is nearest to the field of interest in the dataset. For example the text 'Organization:' is the invariant text label that is nearest to the organization name in HTML document figure at the beginning of this Section. As distance measure we use the length of the shortest path in the document tree (for example the distance of a node to its parent is one; to its sibling, two; to its uncle, three.).

3.1 The k-testable algorithm

Our approach to information extraction using tree automata induction, presented in the previous section, can in principle be tried with any tree automata inference algorithm available. In our prototype implementation, we have chosen one of the more useful and practical algorithms available, namely, the k-testable algorithm [18]. This algorithm is parameterized by a natural number k, and the name comes from the notion of a "k-testable tree language". Informally, a tree language (set of trees) is k-testable if membership of a tree in the language can be determined just by looking at all the subtrees of length k (also intermediate ones). The k-testable algorithm is capable of identifying in the limit any k-testable tree language from positive examples only. Since information extraction typically has a locally testable character, it seems very appropriate to use in this context. The choice of k is performed automatically using cross-validation, choosing the smallest k giving the best results.

For the sake of completeness, we describe the algorithm here. We need the following terminology. Given a tree $t = v(t_1 ... t_m)$, $length(t)$ is the number of edges on the longest path between the root and a leaf. The (singleton) set $r_k(t)$ of root trees of length k is defined as:

$$r_k(v(t_1 ... t_m)) = \begin{cases} v & \text{if } k = 1 \\ v(r_{k-1}(t_1) ... r_{k-1}(t_m)) & \text{otherwise} \end{cases} \quad (1)$$

The set $f_k(t)$ of fork trees of length k is defined as:

$$f_k(v(t_1 ... t_m)) = \bigcup_{j=1}^{m} f_k(t_j) \bigcup \begin{cases} \emptyset & \text{if length}(v(t_1 ... t_m)) < k-1 \\ r_k(v(t_1 ... t_m)) & \text{otherwise} \end{cases} \quad (2)$$

Finally, the set $s_k(t)$ of subtrees of length k is defined as:

$$s_k(v(t_1...t_m)) = \bigcup_{j=1}^{m} s_k(t_j) \bigcup \begin{cases} \emptyset & \text{if length}(v(t_1...t_m)) > k-1 \\ v(t_1...t_m) & \text{otherwise} \end{cases} \quad (3)$$

Example 1. For example, if $t = a(b(a(b,x)),c)$ then $r_2(t) = \{a(b,c)\}$; $f_2(t) = \{a(b,c), b(a), a(b,x)\}$; and $s_2(t) = \{a(b,x), b, x, c\}$.

The procedure to learn the tree automaton [18] is shown below. The algorithm takes as input a set of trees over some ranked alphabet V; these trees serve as positive examples. The output is a tree automaton (V, Q, Δ, F).

Let T be the set of positive examples.
$Q = \emptyset$; $F = \emptyset$; $\Delta = \emptyset$;
For each $t \in T$,
- Let $\mathcal{R} = r_{k-1}(t)$, $\mathcal{F} = f_k(t)$ and $\mathcal{S} = s_{k-1}(t)$.
- $Q = Q \cup \mathcal{R} \cup r_{k-1}(\mathcal{F}) \cup \mathcal{S}$
- $F = F \cup \mathcal{R}$
- for all $v(t_1, \ldots, t_m) \in \mathcal{S}$: $\Delta = \Delta \cup \{\delta_m(v, t_1, \ldots, t_m) = v(t_1, \ldots, t_m)\}$
- for all $v(t_1, \ldots, t_m) \in \mathcal{F}$: $\Delta = \Delta \cup \{\delta_m(v, t_1, \ldots, t_m) = r_{k-1}(v(t_1, \ldots, t_m))\}$

Example 2. Applying the algorithm on the term of Example 1 for $k = 3$, we obtain:

- $\mathcal{R} = r_2(t) = \{a(b,c)\}$, $\mathcal{F} = f_3(t) = \{a(b(a),c), b(a(b,x))\}$ and $\mathcal{S} = s_2(t) = \{a(b,x), b, x, c\}$.
- $Q = \{a(b,c), b(a), a(b,x), b, x, c\}$
- $F = \{a(b,c)\}$
- transitions:
  - $a(b,x) \in \mathcal{S} : \delta_2(a,b,x) = a(b,x)$
  - $b \in \mathcal{S} : \delta_0(b) = b$
  - $x \in \mathcal{S} : \delta_0(x) = x$
  - $c \in \mathcal{S} : \delta_0(c) = c$
  - $a(b(a),c) \in \mathcal{F} : \delta_2(a, b(a), c) = a(b,c)$
  - $b(a(b,x)) \in \mathcal{F} : \delta_1(b, a(b,x)) = b(a)$

With more (and larger) examples, more transitions are created and generalisation occurs: also trees different from the given ones will be labeled with an accepting state (a state from F).

4 Experimental Results

We evaluate the k-testable method on the following semi-structured data sets: a collection of web pages containing people's contact addresses which is called the Internet Address Finder (IAF) database and a collection of web pages about stock quotes which is called the Quote Server (QS) database. There are 10 example documents in each of these datasets. The number of fields to be extracted is respectively 94 (IAF organization), 12 (IAF alt name), 24 (QS date), and 25 (QS vol). The motivation to choose these datasets is as follows. Firstly they are benchmark datasets that are commonly used for research in information extraction, so we can compare the results of our method directly with the results of other methods. Secondly they are the only (online available) datasets that, to the best of our knowledge, require the extraction on the whole node of a tree and not a part of a node. These datasets are available online from RISE.[5]

We use the same criteria that are commonly used in the information retrieval 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 methods: an algorithm based on Hidden Markov Models (HMMs) [10], the Stalker wrapper induction algorithm [16] and BWI [9]. The results of HMM, Stalker and BWI are adopted from [9]. All tests are performed with ten-fold cross validation following the splits used in [9][6]. Each split has 5 documents for training and 5 for testing. We refer to the related work section for a description of these methods.

As we can see from Table 1 our method performs better in most of the test cases than the existing state-of-the-art methods. The only exception is the field date in the Quote Server dataset where BWI performs better. We can also see that the k-testable algorithm always gets 100 percent of precision. Like most algorithms that learn from positives only, k-testable generalises very cautiously, and thus is oriented towards achieving high precision rather than high recall. The use of a tree language instead of a string language, which increases the expressiveness of the hypothesis space, apparently makes it possible in these cases to avoid incorrect generalisations.

5 Discussion and related work

The running time of the k-testable algorithm in Section 3.1 is $O(k\ m\ log\ m)$, where m is the total length of the example trees. The preprocessing consists of parsing, conversion to the binary tree representation (linear in the size of the document) and the manual insertion of the label x. Our prototype implementation was tested on a Pentium 166 Mhz PC. For the two datasets that we test

[5] http://www.isi.edu/~muslea/RISE/

[6] We thank Nicholas Kushmerick for providing us with the datasets used for BWI.

Table 1. Comparison of the results

	IAF - alt. name			*IAF - organization*			*QS - date*			*QS - volume*		
	Prec	*Recall*	*F1*	*Prec*	*Recall*	*F1*	*Prec*	*Recall*	*F1*	*Prec*	*Recall*	*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

above the average training time ranges from less than a second to some seconds for each k learned.

The time complexity of the extraction procedure is $O(n^2)$ where n is the number of nodes in the document. This runtime complexity depends on the number of nodes in the document where each time it has to substitute one of the node with x when running the automaton. For every node in the document tree the automaton has to find a suitable state for the node. With a suitable data structure for indexing the states the find operation on the states can be implemented to run in constant time. In our implementation the learned automata extract the document in seconds including preprocessing using a rudimentary indexing for the find operation.

Doing some additional experiments on various data, we learned that the value of k has a lot of impact on the amount of generalisation: the lower k the more generalisation. On the other hand, when the distance to the distinguishing context is large, then a large k is needed to capture the distinguishing context in the automaton. This may result in a too specific automaton having a low recall. In future we plan to investigate methods to further generalise the obtained automaton.

There have been a lot of methods that have been used for IE problems, some are described in [15, 22]. Many of them learn wrappers based on regular expressions. BWI [9] is basically a boosting approach in which the weak learner learns a simple regular expression with high precision but low recall. Chidlovskii et al. [3] describe an incremental grammar induction approach; their language is based on a subclass of deterministic finite automata that do not contain cyclic patterns. Hsu and Dung [14] learn separators that identify the boundaries of the fields of interest. These separators are described by strings of fixed length in which each symbol is an element of a taxonomy of tokens (with fixed strings on the lowest level and concepts such as punctuation or word at higher levels). The HMM approach in Table 1 was proposed by Freitag and McCallum [10]. They learn a hidden Markov model, solving the problem of estimating probabilities from sparse data using a statistical technique called shrinkage. This model has been shown to achieve state-of-the-art performance on a range of IE tasks. Freitag [7] describes several techniques based on naive-Bayes, two regular language inference algorithms, and their combinations for IE from unstructured texts. His results demonstrate that the combination of grammatical inference techniques with naive-Bayes improves the precision and accuracy of the extrac-

tion. The Stalker algorithm [16] induces extraction rules that are expressed as simple landmark grammars, which are a class of finite automata. Stalker performs extraction guided by a manually built *embedded catalog* tree, which is a tree that describes the structure of fields to be extracted from the documents. WHISK [22] is a system that learns extraction rules with a top-down rule induction technique. The extraction rules of WHISK are based on a form of regular expression patterns.

Compared to our method the methods mentioned above use methods to learn string languages while our method learns a more expressive tree language. Compared to HMMs and BWI our method does not require the manual specification of the windows length for the prefix, suffix and the target fragments. Compared to Stalker and BWI our method does not require the manual specification of the special tokens or landmarks such as ">" or ";". Compared to Stalker our method works directly on document trees without the need for manually building the *embedded catalog* tree.

Despite the above advantages, there are some limitations of our method compared to the other methods. The first is the fact that our method only outputs the whole node seems to limit its application. One way to make our method more applicable is to do two level extraction. The first level extracts the whole node of the tree and the second extracts a part of the node using a string-based method. Secondly, our method works only on structured documents. This is actually a consequence of using tree automata inference. Indeed our method cannot be used for text-based IE, and is not intended for it. Thirdly, our method is slower than the string-based method because it has to parse, convert the document tree and substitute each node with x when extracting the document. Despite these limitations the preliminary results suggest that our method works better in the two structured domains than the more generally applicable string-based IE methods.

WHIRL is a 'soft' logic system that incorporates a notion of textual similarity developed in the information retrieval community. WHIRL has been used to implement some heuristics that are useful for IE in [4]. In this sense WHIRL is not a wrapper induction system but rather a logic system that is programmed with heuristics for recognizing certain types of structure in HTML documents. Hong and Clark [13] propose a technique that uses stochastic context-free grammars to infer a coarse structure of the page and then uses some user specified rules based on regular expressions to do a finer extraction of the page. Sakamoto et al. [21] propose a certain class of wrappers that use the tree structure of HTML documents and propose an algorithm for inducing such wrappers. They identify a field with a path from root to leaf, imposing conditions on each node in the path that relate to its label and its relative position among siblings with the same label (e.g., "2nd child with label `<B>`"). Their hypothesis language corresponds to a subset of tree automata.

Besides the k-testable algorithm proposed in this paper, we have also experimented with Sakakibara's reversible tree algorithm [19]. Preliminary results with this algorithm suggested that it generalises insufficiently on our data sets, which is why we did not pursue this direction further.

6 Conclusion

We have motivated and presented a novel method that uses tree automata induction for information extraction from structured documents. We have also demonstrated on two datasets that our method performs better in most cases than the string-based methods that have been applied on those datasets. These results suggest that it is worthwhile to exploit the tree structure when performing IE tasks on structured documents.

As future work we plan to test the feasibility of our method for more general IE tasks on XML documents. Indeed, until now we have only performed experiments on standard benchmark IE tasks that can also be performed by the previous string-based approaches, as discussed in the two previous sections. However, there are tasks that seem clearly beyond the reach of string-based approaches, such as extracting the second item from a list of items, where every item itself may have a complex substructure. Of course, experimental validation remains to be performed. Interestingly, recent work by Gottlob and Koch [12] shows that all existing wrapper languages for structured document IE can be captured using tree automata, which strongly justifies our approach.

Other directions to explore are to incorporate probabilistic inference; to infer unranked tree automata formalisms directly; and to combine unstructured text extraction with structured document extraction.

Acknowledgements

We thank anonymous reviewers for their helpful feedbacks. 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

1. D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.
2. 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 Innovative Applications of Artificial Intelligence*, pages 328–334. AAAI Press / The MIT Press, 1999.
3. B. Chidlovskii, J. Ragetli, and M. de Rijke. Wrapper generation via grammar induction. In *11th European Conference on Machine Learning, ECML'00*, pages 96–108, 2000.
4. W. W. Cohen. Recognizing structure in web pages using similarity queries. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on on Innovative Applications of Artificial Intelligence*, pages 59–66, 1999.
5. 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.
6. J. Cowie and W. Lehnert. Information extraction. *Communications of the ACM*, 39(1):80–91, 1996.

7. D. Freitag. Using grammatical inference to improve precision in information extraction. In *ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition*, 1997.
8. D. Freitag. Information extraction from HTML: Application of a general learning approach. In *Proceedings of the Fifteenth Conference on Artificial Intelligence AAAI-98*, pages 517–523, 1998.
9. D. Freitag and N. Kushmerick. Boosted wrapper induction. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Innovative Applications of AI Conference*, pages 577–583. AAAI Press, 2000.
10. D. Freitag and A. McCallum. Information extraction with HMMs and shrinkage. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.
11. E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.
12. G. Gottlob and K. Koch. Monadic datalog over trees and the expressive power of languages for web information extraction. In *21st ACM Symposium on Principles of Database Systems*, June 2002. To appear.
13. T. W. Hong and K. L. Clark. Using grammatical inference to automate information extraction from the web. In *Principles of Data Mining and Knowledge Discovery*, pages 216–227, 2001.
14. 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.
15. I. Muslea. Extraction patterns for information extraction tasks: A survey. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.
16. I. Muslea, S. Minton, and C. Knoblock. Hierarchical wrapper induction for semi-structured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 4:93–114, 2001.
17. C. Pair and A. Quere. Définition et etude des bilangages réguliers. *Information and Control*, 13(6):565–593, 1968.
18. J. Rico-Juan, J. Calera-Rubio, and R. Carrasco. Probabilistic k-testable tree-languages. In A. Oliveira, editor, *Proceedings of 5th International Colloquium, ICGI 2000, Lisbon (Portugal)*, volume 1891 of *Lecture Notes in Computer Science*, pages 221–228. Springer, 2000.
19. Y. Sakakibara. Efficient learning of context-free grammars from positive structural examples. *Information and Computation*, 97(1):23–60, 1992.
20. Y. Sakakibara. Recent advances of grammatical inference. *Theoretical Computer Science*, 185(1):15–45, 1997.
21. H. Sakamoto, H. Arimura, and S. Arikawa. Knowledge discovery from semistructured texts. In S. Arikawa and A. Shinohara, editors, *Progress in Discovery Science - Final Report of the Japanese Discovery Science Project*, volume 2281 of *LNAI*, pages 586–599. Springer, 2002.
22. S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.
23. M. Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Information and Control*, 27:1–36, 1975.
24. L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.
25. Extensible markup language (XML) 1.0 (second edition). W3C Recommendation 6 October 2000. `www.w3.org`.