

# Learning (k,l)-Contextual Tree Languages for Information Extraction

*Stefan Raeymaekers  
Maurice Bruynooghe  
Jan Van den Bussche*

*Report CW 390, September 2004*



Katholieke Universiteit Leuven  
Department of Computer Science  
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# Learning $(k,l)$ -Contextual Tree Languages for Information Extraction

*Stefan Raeymaekers*  
*Maurice Bruynooghe*  
*Jan Van den Bussche*

*Report CW 390, September 2004*

Department of Computer Science, K.U.Leuven

## **Abstract**

Learning regular languages from positive examples only is known to be infeasible. A common solution is to define a learnable subclass of the regular languages. In the past, this has been done for regular string languages. Using ideas from those techniques, we define a learnable subclass of regular unranked tree languages, called the  $(k,l)$ -contextual tree languages. We describe the use of this subclass to induce wrappers for Information Extraction from structured documents, such as web pages. Experiments show that our algorithm is able to learn from very few data, and compares favorably to similar state of the art approaches.

# Learning $(k,l)$ -Contextual Tree Languages for Information Extraction

Stefan Raeymaekers<sup>1</sup>, Maurice Bruynooghe<sup>1</sup>, and Jan Van den Bussche<sup>2</sup>

<sup>1</sup> K.U.Leuven, Dept. of Computer Science, Celestijnenlaan 200A, B-3001 Leuven,  
{stefanr,maurice}@cs.kuleuven.ac.be

<sup>2</sup> University of Limburg, Dept. WNI, Universitaire Campus, B-3590 Diepenbeek,  
jan.vandenbussche@luc.ac.be

**Abstract.** Learning regular languages from positive examples only is known to be infeasible. A common solution is to define a learnable subclass of the regular languages. In the past, this has been done for regular string languages. Using ideas from those techniques, we define a learnable subclass of regular unranked tree languages, called the  $(k,l)$ -contextual tree languages. We describe the use of this subclass to induce wrappers for Information Extraction from structured documents, such as web pages. Experiments show that our algorithm is able to learn from very few data, and compares favorably to similar state of the art approaches.

## 1 Introduction

The class of regular languages is not learnable from positive examples only [8]. One solution for this negative result is defining learnable subclasses. Examples of this approach in the case of string languages are  $k$ -reversible languages[2],  $k$ -contextual languages[14] and  $k$ -testable languages[7]. Since  $k$ -contextual and  $k$ -testable languages are equivalent[1], we will refer to them as  $k$ -local languages. In the case of tree languages,  $k$ -testable tree languages are introduced in [6, 9], as an extension of the  $k$ -contextual and  $k$ -testable string languages, with probabilistic extensions in [18], and in [10] local unranked tree automata are introduced. In Section 3 we define a new subclass for the class of regular unranked tree languages, called  $(k,l)$ -contextual tree languages and we present an inference algorithm for it. The class of  $(k,l)$ -contextual tree languages is an extension of the  $k$ -contextual string languages [14, 1].

Information extraction(IE) from structured documents (HTML, XML), aims at extracting specific information from structurally similar documents. Often it is referred to as *wrapper induction*. Some examples can be found in [12, 15, 20, 3, 10, 11]. In [10] tree automata are learned from positive examples only. Each example is a document, in which one of the elements that is desired, is marked with a special marker. When used to extract information from a document with  $n$  targets,  $n$  runs with the learned automaton are performed, where one candidate target is marked in each run; the targeted element is extracted when the automaton accepts the marked document. In Section 4, we use the same approach, using a

$(k,l)$ -contextual grammar to represent the wrapper. We present a new algorithm specially tailored towards this application. We compare its expressiveness with the approach in [10].

Both the algorithm in [10] and our algorithm suffer from the drawback that they need parameters. While one parameter suffices for the algorithm in [10], our algorithm needs two. The solution suggested in [10] is to choose the parameter that yields the best precision and recall after cross-validation on the training set. The training and test sets used in [10], to compare their algorithm to some string based algorithms [4, 16, 5] are the same as in [5]. Each training set is made up of 5 documents, with all the desired elements marked. Resulting in some hundreds of examples. One can argue that this goes beyond learning from positive examples only. Indeed, when the document is completely marked, it is easy to infer negative data by marking in a document an element that is not marked in any of the positive examples. When 'true' positive training sets are used, precision and recall are undefined and the suggested solution is not valid. In Section 5 we suggest a heuristic algorithm that yields reasonable results.

Examples in an IE task have to be marked manually. Therefore induction algorithms that produce good results with very few examples are preferable. In Section 6 we show the results of testing and comparing the two algorithms on very small training sets.

Some preliminary definitions are provided in Section 2 and we conclude in Section 7.

## 2 Preliminary Definitions

We define the alphabet  $\Sigma$  as a finite set of symbols. The set of all unranked trees over  $\Sigma$  is defined as  $T(\Sigma) = \{f(s) \mid f \in \Sigma, s \in T(\Sigma)^*\}$ . We usually denote  $f(\epsilon)$ , where  $\epsilon$  is the empty sequence, by  $f$ . A *tree language* is any subset of  $T(\Sigma)$ . The set of  $(k,l)$ -roots of a tree  $f(t_1, \dots, t_n)$  is the singleton  $\{f\}$  if  $l=1$ , otherwise it is the set of trees obtained by extending the root  $f$  with  $k$   $(k,l-1)$ -roots taken from successive children of  $f$  (all children if  $k < n$ ).

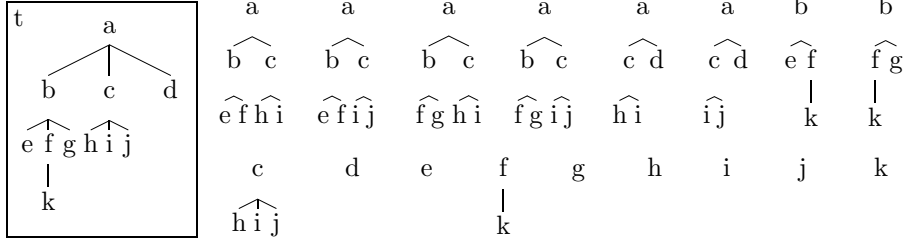
Formally (using  $r_{k,l}(t)$  to denote the elements of  $R_{k,l}(t)$ ):

$$R_{(k,l)}(f(t_1 \dots t_n)) = \begin{cases} \{f\} & \text{if } l = 1 \\ \{f(R_{(k,l-1)}(t_1) \dots R_{(k,l-1)}(t_n))\} & \text{if } l > 1 \text{ and } k > n \\ \{f(R_{(k,l-1)}(t_p) \dots R_{(k,l-1)}(t_{p+k-1})) \mid p = 1 \dots n-k+1\} & \text{otherwise} \end{cases}$$

Finally, a  $(k,l)$ -root of a subtree of  $t$  is a  $(k,l)$ -fork of  $t$ ; hence the set of  $(k,l)$ -forks is given by:

$$F_{(k,l)}(f(t_1 \dots t_n)) = \{t \mid t \in R_{(k,l)}(f(t_1 \dots t_n))\} \cup_{i=1}^n F_{(k,l)}(t_i)$$

*Example 1.* Below we show graphically the  $(2,3)$ -forks of a tree  $t$ . The first 6 of these forks, are the  $(2,3)$ -roots of  $t$ .



### 3 (k,l)-Contextual Languages

The definitions of  $k$ -local languages[14, 7, 1] and their extension  $k$ -testable tree languages[6, 9] incorporate some way to distinguish the initial  $k$ -grams/forks and the final  $k$ -grams/forks within all the  $k$ -grams/forks of a set of strings/trees. One possible way is to use 3 sets in the definition, a separate set for the initial and for the final elements. Another way is to indicate the beginning and end of a string with a special marker, that does not belong to the alphabet. Since  $(k,l)$ -forks can be initial and final in both horizontal and vertical direction, the first way results in a complex definition. We have chosen for the second way. In [1], the definition of  $k$ -grams is such that the special marker is added automatically. We have opted for a simple definition of  $(k,l)$ -forks and we add the marker in a preprocessing step, by performing a transformation HV on the tree.

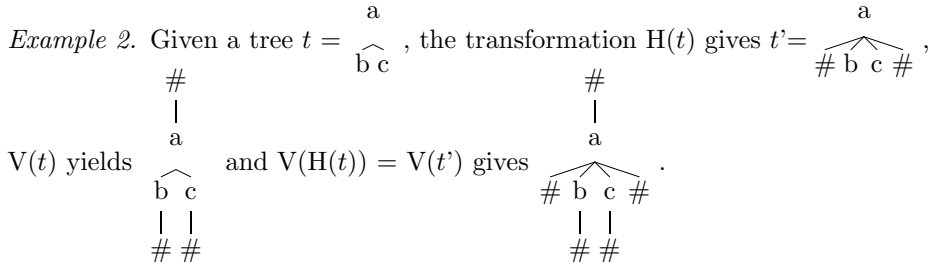
Given a tree  $t \in T(\Sigma)$ , the transformation HV is defined as  $HV(t) = V(H(t))$ . Where H and V are the horizontal and the vertical transformation. These are defined respectively as

$$\begin{aligned} H(f) &= f \\ H(f(t_1 \dots t_n)) &= f(\# H(t_1) \dots H(t_n) \#) \end{aligned}$$

and

$$\begin{aligned} V(t) &= \#(V'(t)) \\ V'(f) &= f(\#) \\ V'(f(t_1 \dots t_n)) &= f(V'(t_1) \dots V'(t_n)) \\ V'(\#) &= \# \end{aligned}$$

with  $f \in \Sigma$ ,  $\# \notin \Sigma$  and  $t_1 \dots t_n \in T(\Sigma \cup \{\#\})$ .



Let  $G$  be a set of  $(k,l)$ -forks over an alphabet  $\Sigma \cup \{\#\}$ , with the property that  $\#$  only occurs as a leaf or as a root node with exactly one child. The  $(k,l)$ -contextual tree language based on  $G$  is defined as

$$L(G) = \{t \in T(\Sigma) \mid F_{(k,l)}(HV(t)) \subseteq G\}.$$

The  $k$ -testable languages are defined by a set of forbidden substrings, while the  $k$ -contextual languages are defined by a set of allowed substrings. Our definition states the set of allowed subforks, like in the latter, hence the name  $(k,l)$ -contextual tree languages.

We remark that when we mention  $k$ -testable languages we mean locally testable in the strict sense, and not the more general definition given in [13]. In this notion,  $G$  will be a set of sets of forks, while the language defined by  $G$  will contain every tree, whose forks are a subset of at least one of the sets of forks in  $G$ . This Approach is more expressive, but needs more examples to learn. Given that experiments (Section 6) show that our approach is already sufficiently expressive for the data sets at hand, we leave this approach for later study.

The  $k$ -testable tree languages[6, 9] are a subset of the  $(k,l)$ -contextual tree languages. They are equivalent to the  $(\infty,k)$ -contextual tree languages. Note that the  $k$  in  $k$ -testable tree languages indicates the vertical direction, which is  $l$  in our definition. The local unranked tree automata[10] are based on  $(k,1)$ -contextual tree languages. The lack of expressiveness in the vertical direction of this subset is remedied with some extra preprocessing (see Section 4.3)

The inference algorithm for  $(k,l)$ -contextual tree languages (Algorithm 1) simply collects all forks. Checking for membership of a tree  $t$  (Algorithm 2) is done by checking whether the forks of  $t$  are a subset of the forks defining the  $(k,l)$ -contextual tree language.

---

**Algorithm 1** Inference algorithm for  $(k,l)$ -contextual languages

---

**Input:** A finite set of trees  $T$ , and positive integers  $k$  and  $l$ .

**Output:** A set  $G$  of  $(k,l)$ -forks for language  $L$ , such that  $L$  is the smallest  $(k,l)$ -contextual tree language that contains  $T$ .

- 1:  $G = \emptyset$
  - 2: **for** each  $t \in T$  **do**
  - 3:    $t' = HV(t)$
  - 4:    $G = G \cup (k,l)$ - forks from  $t'$
  - 5: **end for**
- 

---

**Algorithm 2** Membership algorithm for  $(k,l)$ -contextual languages

---

**Input:** A tree  $t$  and a set  $G$  of  $(k,l)$ -forks.

**Output:** A boolean value *contains* that indicates whether  $t \in L(G)$ .

- 1:  $t' = HV(t)$
  - 2:  $F = (k,l)$ - forks from  $t'$
  - 3: *contains* =  $F \subseteq G$
-

## 4 Extraction Application

This section describes wrapper induction as an application of  $(k,l)$ -contextual tree language inference. We follow the same approach as [10, 11], which we explain briefly in Section 4.1. In Section 4.2 some modifications to our inference algorithm are presented. This modifications aim to take advantage of the properties of this application. In Section 4.3 we discuss the expressiveness of our method and compare it to the system based on local unranked tree automata[10].

### 4.1 Approach

We describe here generally the incremental learning process (Algorithm 3) and the extraction process (Algorithm 4). Any incremental string or tree language inference algorithm can be used. An *example* for the learning task consists of an HTML page that has one of its requested text nodes<sup>3</sup> marked. The extraction algorithm receives an unmarked page, but during the iteration over the nodes, the page is processed with exactly one node marked.

---

**Algorithm 3** Wrapper Induction

---

**Input:** A set of examples  $E$ , and parameters  $P$  for the inference algorithm.

**Output:** A wrapper  $W$  and a set of contexts  $C$ .

```
1:  $C = \text{deriveContexts}(E)$ 
2:  $W = \text{initialWrapper}$  //depends on inference algorithm
3: for each example  $e$  in  $E$  do
4:    $e' = \text{convert}(e, C)$ 
5:    $W = \text{update}(W, e')$  //depends on inference algorithm
6: end for
```

---

---

**Algorithm 4** Extraction

---

**Input:** A HTML page  $p$ , a wrapper  $W$  and a set of contexts  $C$ .

**Output:** A list  $L$  of extracted nodes.

```
1: for each text node  $n$  in  $p$  do
2:    $\text{mark}(n)$ 
3:    $p' = \text{convert}(p, C)$ 
4:   if  $\text{isMemberOf}(p', W)$  then
5:     add  $n$  to list  $L$ 
6:   end if
7:    $\text{unmark}(n)$ 
8: end for
```

---

The *convert* operation is the same for both induction and extraction. This operation replaces the text in every text node by '@', if that text does not belong

<sup>3</sup> In HTML, the text nodes are always leaf nodes.

to the set of contexts. This preprocessing, enables the system to generalize over the infinite alphabet of all possible texts. A context is a string that is crucial to distinguish the requested nodes. Often the latter nodes can be distinguished by the structure alone. In that case the set of contexts can be empty. The set of contexts is learned automatically from the training set. We use the same algorithm as in [10, 11]. This algorithm searches strings that occur at the same distance from each marked node in the training set. The distance we use is defined such that the distance from a node to its parent is one; to its sibling two; to its uncle three. The algorithm returns a set with one context, the context that occurs at the smallest distance from the marked nodes.

The implementation of the functions *initialWrapper*, *update* and *isMemberOf* depends on the choice of a basic inference algorithm. Given  $(k,l)$ -contextual tree languages, we could use Algorithm 2 for the *isMemberOf* function, and split Algorithm 1 to make it incremental. The function *initialWrapper* is substituted by Line 1, and for function *update*, we use Lines 3-4. Although this implementation works, we will improve it in Section 4.2, by adapting the algorithms for HTML pages.

## 4.2 Modifications

Given that the documents are HTML pages, we know that the root node has always a `<html>`-tag, and the leaves are always text nodes or some specific subset of HTML tags that can occur without children. Hence the initial and final nodes in the vertical direction are distinguishable and we can omit the V-transformation.

Another observation is that in the approach for information extraction, the set of forks splits naturally in two: on one hand, the forks that contain a marker, and on the other hand, the forks without a marker. The first set describes the local structure of the marked node, while the second describes the general structure of the document. When some marked document gets rejected, the reason is either that it is marked incorrectly or that it does not belong in the domain of pages for which the wrapper is trained. The former is checked by the set of marked forks checks, while the latter is checked by the set of unmarked forks. We assume that the input for the extraction algorithm will all be from the correct domain, therefore if a page gets rejected due to the unmarked forks, this means that the wrapper did not see enough examples, or did not generalize enough, to identify the domain correctly. This could be solved by generalizing more over the unmarked forks, but because we assume that the page does already belong to the correct domain, the need to check whether the page belongs to the domain disappears. Consequently, we filter out the unmarked forks and consider only the marked forks.

Given that we consider only the local structure around the marked node, we choose to also omit the H-transformation. This implies that we lose a bit of expressiveness (it might be that knowledge of initial or final nodes in the horizontal direction is important to describe the local structure), but we gain a bit in generalizing power (when the requested nodes occur in some list or table

structure, we do not need separate examples for the beginning, middle and end of the list). This choice was validated by some early experiments.

Summarized, we made the following modifications: the whole VH-transformation is omitted and when the forks are extracted from an example or a query, we consider only the marked forks.

### 4.3 Expressiveness

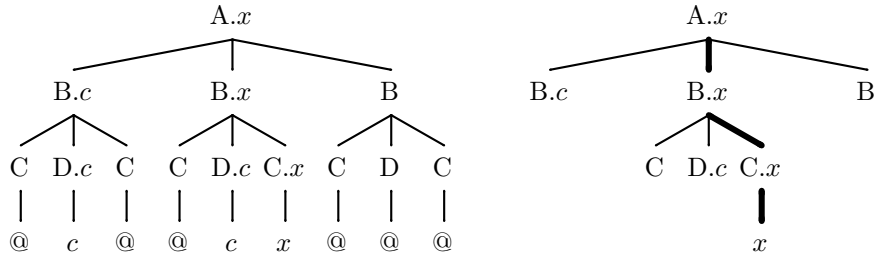
Below we will first briefly explain the method of [10], before we compare its expressivity with the method proposed in this paper.

The same approach is used as described in Section 4.1. This means that a text node in a preprocessed tree is either  $x$  (the single marked node),  $c$  (the one possible context),  $@$  (all other text nodes). The method basically infers a  $(k,l)$ -contextual language, but both example and query trees get an extra transformation that alters the expressiveness of the language. This transformation consists of two separate transformations.

The first transformation changes every node  $f$  into a node  $f.x$ , if its subtree contains the  $x$ -node. And changes it into a node  $f.c$ , if its subtree contains a  $c$ -node, and does not contain the  $x$ -node. Hence, limited information is passed infinitely upwards, making the method not purely local. This is not a problem as long as the resulting subclass stays inferable [8]. This transformation enhances the expressiveness of the original language.

The second transformation is in [10] actually part of the algorithm, but we find it easier to explain it as another separate transformation. This transformation cuts the tree, such that only the path from the root to the  $x$ -node is left, together with the siblings of the nodes on that path. In this way the parts farther away from the marked node are ignored. This enhances the generalizing power of the resulting language (and reduces the expressiveness).

*Example 3.* The left tree below shows a tree after the first transformation, while the tree on the right shows the result of applying the second transformation to that same tree.



Concerning the VH-transformation defined in the definition of  $(k,l)$ -contextual languages (Section 3), the V-transformation is also omitted in the algorithm in [10], but the H-transformation is retained.

When we compare our algorithm(KL) with the one in [10](K), we see that thanks to the first transformation, K, based on  $(k,1)$ -contextual tree languages,

can express some global vertical relations, while KL is purely local. This means that KL can not describe the relation between  $z$  and an ancestor that can be an arbitrary number of levels higher. Although we could construct a synthetical dataset for which this is needed, we have not encountered any such data set in our experiments. For paths of the tree that do not contain the  $x$ -node or a  $c$ -node, the expressivity of K stays  $(k,1)$ -contextual.

The second transformation in K implies a loss of expressiveness. The structure of sibling nodes of the target can no longer be distinguished. In KL this can be distinguished within the  $(k,1)$  neighbourhood. We encountered several examples of pages where this would be needed for correct extraction. For example a page containing a table with descriptions of bargains from which one would like to extract only those with a picture of the item. The picture, when present, occupies the first cell of the row, which is a sibling of the target cell.

## 5 Parameter Estimation

For training sets with only positive examples, it is hard to find the optimal parameters, since no feedback over precision and recall is available. We devised a heuristic algorithm that estimates the optimal value of the parameters, for a given training set. This heuristic is based on the evolution of the number of extractions over the  $(k,l)$ -parameter space.

We make the assumption that the nodes required in the extraction task can be defined locally. We already made this assumption when we choose to use  $(k,1)$ -contextual tree languages. The  $k$  and  $l$  parameters define a kind of local window. The smallest window ( $k=1, l=0$ ) will extract everything. When the  $(k,l)$ -window is enlarged, more and more of the surrounding features are taken into account, and less nodes are extracted. At one point the window is big enough to catch all the necessary structure features, to distinguish the required nodes correctly. At another point, the window becomes too big, and details of the enclosing structure can lead to overspecialization, and the number of extractions goes further down. Often there is some leeway between these two points, where the number of extractions stays constant. This is detected by the heuristic.

Due to space restrictions we can not describe this approach in detail here. We present this algorithm in [17].

## 6 Experiments

We evaluate our approach on the WIEN data sets (available at the RISE repository[19]). Some of the sets are discarded since they do not include labels. The other sets are split in different tasks: a task aiming at the extraction of a  $n$ -tuple is splitted in  $n$  extraction tasks. We refer to each field extraction task, with the name of the original data set together with the index of the field in the tuple. From these tasks we keep only those that extract a complete node, resulting in a total number of 36 extraction tasks.

We used the metrics precision(P), recall(R) and F1 score. Given E, the number of text nodes extracted from the test set, C, the number of correctly extracted text nodes, and T, the total number of text nodes in the test set. We can define these metrics as  $P=C/E$ ,  $R=C/T$ , and  $F1=2PR/(P+R)$ , the harmonic mean of P and R.

A first experiment evaluated the expressiveness of the languages. We used the complete data set for learning and checked that extraction did not return more fields than those marked during the learning. For both algorithms (the algorithm in [10], and ours) and all data sets, we found a parameter setting that resulted in a F1 score of 100%. Hence both approaches are expressive enough for the extraction tasks at hand. This means that when no parameter setting can be found such that an algorithm does yield 100% F1 score on a smaller data set, this is only due to the fact that there are not enough examples in the training set for that algorithm to learn optimally. And not because of lack of expressiveness.

In a second experiment we use as training set, a set of 5 examples, randomly picked from the data set. The other examples are used as test set. We perform each experiment 5 times, each time with a different set of random examples. This results in  $36 \times 5$  experiments. The results of this experiment are shown in Table 1. Each value is the mean over the 5 different random experiments. The variance over the different experiments was low. In most cases when a mean does not reach 100%, all the experiments do not reach 100%. The first set of results is obtained with the algorithm based on local unranked automata[10]. The parameter  $k$  is chosen optimally over the test set. The second set of results are the results of our new algorithm, with the parameters  $k$  and  $l$  chosen optimally over the test set. The last set of results is also obtained with our new algorithm, but the parameters are estimated with the heuristic from Section 5. The column *ctx* indicates if the context algorithm was used (for all of the algorithms).

When we compare the results of our algorithm with the one in [10], we see that for the 36 tasks only 2 have a worse F1 score, while 24 have a better F1 score. Looking at the results from the estimated parameters, we see that the optimal parameters were chosen in 26 tasks. When we compare our algorithm with estimated parameters, to the algorithm in [10] with optimal parameters, the F1 scores are better in 19 tasks, and worse in only 6 tasks.

In a third experiment, we use 1 random example for each data set. Our algorithm is used and the parameters are again obtained by searching the optimal ones for the entire data set. Note that for the sets that needed a context in the previous experiment, we pass the correct context ourselves, since the algorithm we use to determine the context, needs at least 2 examples. We see that for 31 of the 36 tasks the evaluation on the test set yields a 100% F1 score.

## 7 Conclusion

We have introduced a new subclass of the regular tree languages, called  $(k,l)$ -contextual tree languages, that is learnable from positive examples only. This is an extension from  $k$ -contextual languages, defined as a subclass of the regular

**Table 1.** Results for data sets with 5 examples

Data set	ctx	local: optimal k			optimal k,l			estimated k,l		
		Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
s1-1		100.0	81.0	89.1	100.0	100.0	100.0	94.4	100.0	97.1
s1-3		97.5	85.3	90.4	100.0	97.5	98.7	52.6	100.0	69.0
s1-4		100.0	66.4	78.8	100.0	100.0	100.0	100.0	100.0	100.0
s3-2		100.0	95.4	97.6	100.0	100.0	100.0	100.0	100.0	100.0
s3-3		100.0	96.6	98.2	100.0	100.0	100.0	100.0	100.0	100.0
s4-1		100.0	84.6	91.6	100.0	100.0	100.0	100.0	100.0	100.0
s5-2		100.0	88.6	93.8	98.0	100.0	98.9	90.0	100.0	94.7
s8-2		100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s8-3		100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s10-2		100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s10-4		100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s12-2		100.0	96.9	98.4	96.4	98.2	97.8	100.0	8.6	15.9
s13-2		100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s13-4		100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s14-3		100.0	99.0	99.5	100.0	100.0	100.0	100.0	100.0	100.0
s15-2		100.0	94.4	97.1	100.0	100.0	100.0	100.0	100.0	100.0
s19-4		100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
s20-3		100.0	96.9	98.5	100.0	100.0	100.0	100.0	100.0	100.0
s20-4		100.0	95.1	97.5	100.0	100.0	100.0	33.3	100.0	50.0
s20-5		100.0	95.1	97.5	100.0	100.0	100.0	33.3	100.0	50.0
s20-6		100.0	96.9	98.5	100.0	100.0	100.0	100.0	100.0	100.0
s22-2		100.0	87.9	93.3	100.0	100.0	100.0	100.0	99.4	99.7
s23-1		100.0	95.5	97.6	100.0	100.0	100.0	100.0	100.0	100.0
s23-3		100.0	89.6	94.4	100.0	100.0	100.0	100.0	100.0	100.0
s25-2		100.0	94.5	97.2	100.0	100.0	100.0	100.0	100.0	100.0
s30-2		100.0	93.3	96.0	100.0	100.0	100.0	100.0	93.3	96.0
iaf-1	✓	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
iaf-2	✓	100.0	89.4	93.9	100.0	100.0	100.0	100.0	100.0	100.0
qs-1		100.0	94.2	96.6	100.0	94.2	96.6	100.0	94.2	96.6
qs-2		100.0	100.0	100.0	100.0	78.9	87.8	100.0	78.9	87.8
bigbook-2		100.0	89.4	94.3	100.0	100.0	100.0	94.8	100.0	97.3
bigbook-3		100.0	78.7	88.0	100.0	100.0	100.0	100.0	100.0	100.0
okra-1	✓	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
okra-2	✓	100.0	98.6	99.3	100.0	100.0	100.0	100.0	100.0	100.0
okra-3	✓	100.0	98.2	99.1	100.0	100.0	100.0	100.0	95.1	97.4
okra-4	✓	100.0	98.2	99.1	100.0	100.0	100.0	100.0	100.0	100.0

string languages, to tree languages. We presented an inference algorithm for this new class.

We showed that this new class of languages can be used in IE from web pages (structured documents). We compared the expressiveness of our, for this application modified algorithm, to a state of the art algorithm. Experimentally we show that our algorithm gives very good results from few examples, often only one example suffices. It performs better compared to a state of the art algorithm that uses a similar approach.

The good results concerning the expressiveness indicate that often an IE task for web pages can be defined locally.

## References

1. Helena Ahonen. *Generating grammars for structured documents using grammatical inference methods*. PhD thesis, University of Helsinki, Department of Computer Science, 1996.
2. Dana Angluin. Inference of reversible languages. *Journal of the ACM (JACM)*, 29(3):741–765, 1982.
3. Boris Chidlovskii, Jon Ragetli, and Maarten de Rijke. Wrapper generation via grammar induction. In *Proc. 11th European Conference on Machine Learning (ECML)*, volume 1810, pages 96–108. Springer, Berlin, 2000.
4. D. Freitag and A. McCallum. Information extraction with HMMs and shrinkage. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.
5. Dayne Freitag and Nicholas 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.
6. Pedro García. Learning  $k$ -testable tree sets from positive data. Technical report, Technical Report DSIC-ii-1993-46, DSIC, Universidad Politecnica de Valencia, 1993.
7. Pedro García and Enrique Vidal. Inference of  $k$ -testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(9):920–925, 1990.
8. E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.
9. Timo Knuutila. Inference of  $k$ -testable tree languages. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition: Proc. of the Intl. Workshop*, pages 109–120, Singapore, 1993. World Scientific.
10. Raymondus Kosala, Maurice Bruynooghe, Hendrik Blockeel, and Jan Van den Bussche. Information extraction from web documents based on local unranked tree automaton inference. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 403–408. Morgan Kaufmann, 2003.
11. Raymondus Kosala, Jan Van den Bussche, Maurice Bruynooghe, and Hendrik Blockeel. Information extraction in structured documents using tree automata induction. In *PKDD*, volume 2431 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2002.
12. Nickolas Kushmerick, Daniel S. Weld, and Robert B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

13. Robert McNaughton. Algebraic decision procedures for local testability. *Math. Systems Theory*, 8(1):60–76, 1974.
14. S. Muggleton. *Inductive Acquisition of Expert Knowledge*. Addison-Wesley, 1990.
15. Ion Muslea, Steve Minton, and Craig Knoblock. A hierarchical approach to wrapper induction. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 190–197, Seattle, WA, USA, 1999. ACM Press.
16. Ion Muslea, Steve Minton, and Craig Knoblock. Hierarchical wrapper induction for semistructured information sources. *Journal of Autonomous Agents and Multi-Agent Systems*, 4:93–114, 2001.
17. Stefan Raeymaekers and Maurice Bruynooghe. Parameterless information extraction using (k,l)-contextual tree languages. In *BNAIC 2004 - Proceedings of the 16th Belgian-Dutch Conference on Artificial Intelligence*, 2004.
18. Juan Ramón Rico-Juan, Jorge Calera-Rubio, and Rafael C. Carrasco. Probabilistic k-testable treelanguages. In A.L. Oliveira, editor, *Proceedings of 5th International Colloquium, ICGI*, pages 221–228, 2000.
19. Rise (1998). a repository of online information sources used in information extraction tasks. [<http://www.isi.edu/info-agents/RISE/index.html>]. University of Southern California, Information Sciences Institute.
20. Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.