Using a More Powerful Teacher to Reduce the Number of Queries of the L* Algorithm in Practical Applications

André L. Martins[1], H. Sofia Pinto[2], and Arlindo L. Oliveira[3]

INESC-ID/IST - Av. Alves Redol, 9. 1000-029 Lisboa, Portugal
almar@algos.inesc-id.pt[1], sofia@algos.inesc-id.pt[2], aml@inesc-id.pt[3]

Abstract. In this work we propose to use a more powerful teacher to effectively apply query learning algorithms to identify regular languages in practical, real-world problems. More specifically, we define a more powerful set of replies to the membership queries posed by the L* algorithm that reduces the number of such queries by several orders of magnitude in a practical application. The basic idea is to avoid the needless repetition of membership queries in cases where the reply will be negative as long as a particular condition is met by the string in the membership query. We present an example of the application of this method to a real problem, that of inferring a grammar for the structure of technical articles.

1 Introduction and Motivation

Learning using feedback from the teacher, also known as active learning, is an important area of research, with many practical applications. One of the best known approaches to apply active learning to the inference of sequential models is the L* algorithm.

In this work we describe an improvement to the L* algorithm, that strongly reduces the number of queries, in a practical application in the area of ontology learning.

Ontologies provide a shared and common understanding of a domain that can be communicated between people, as well as between heterogeneous and widely spread application systems. Typically, ontologies are composed of a set of terms representing concepts (hierarchically organized) and some specification of their meaning. This specification is usually achieved by a set of constraints that restrict the way those terms can be combined. The latter set constrains the semantics of a term, since it restricts the number of possible interpretations of the term. Therefore, we can use an ontology to represent the semantic meaning of given terms.

In our case, we are interested in learning an ontology about articles. Using the semantic information, provided by the ontology, search engines can focus on the relevant parts of documents, therefore improving precision and recall.

Our starting point is a small set of 13 basic concepts: title, author(s), abstract title, abstract text, section title, simple text, formated text, subsection title, Figure caption, etc. As an intermediate step towards learning the ontology, we aim at inferring a description of an automaton that encodes the structure of articles. For that we have used query learning. However, existing algorithms for query learning use an exceedingly large number of queries for problems of reasonable size, a feature that makes them unusable in real world settings. In this work we propose a solution to the large number of queries required by the L* algorithm in this and other practical settings.

The remainder of this paper is organized as follows: first, we briefly describe the problem and related work (Sect. 2). We describe, in Sect. 3, the algorithm for query learning that is our starting point, L* . We then describe the approach we used to solve the main problem found when using the L* algorithm, the large number of membership queries that needs to be answered (Sect. 4). Our results are presented and discussed in Sect. 5. We end with conclusions (Sect. 6).

2 Related Work

This work addresses the problem of inferring a regular language using queries and counter-examples. The problem of regular language learning has been extensively studied, both from a practical and theoretical point of view.

Selecting the minimum deterministic finite automaton (DFA) consistent with a set of pre-defined, labeled, strings is known to be NP-complete [1]. Furthermore, even the problem of finding a DFA with a number of states only polynomially larger than the number of states of the minimum solution is also NP-complete [2].

Fortunately, the problem becomes easier if the algorithm is allowed to make queries or to experiment with the unknown automaton. Angluin proposed the L* algorithm [3], a method based on the approach described by Gold [4], that solves the problem in polynomial time by allowing the algorithm to ask membership queries. Schapire [5] proposes an interesting alternative approach that does not require the availability of a reset signal to take the automaton to a known state.

Our work aims at making the L* algorithm more applicable to real world problems, by using a more powerful teacher.

We have chosen a particular problem to apply the proposed approach, that of inferring a regular language that models the structure of a document (in our case, of a technical article). Such a model can later be used to label the different parts of a document, a first step towards the desired goal of semantically labeling the document.

A number of approaches have been proposed to date to the problem of inferring the structure of a document using language models. Additionally, a number of methods have been proposed to efficiently detect structures in sequences of symbols using languages (regular or context-free) as models [6,7,8]. However, these approaches are generally concerned with the identification of repeating structures and not, necessarily, semantic units.

We believe that the application of a grammatical inference method to the inference of semantic units in documents will only lead to interesting results if a human teacher is involved in the learning process. Automatic learning of semantic structures from (possibly labeled) sets of documents is a worthwhile goal, but is likely to require very large corpora and efficient algorithms that do not yet exist.

Other approaches that do not use languages as models for the text also exist, but are more limited in their potential scope, since they look for local features of the text, such as fonts and formats [9,10,11,12,13]. The techniques used in these approaches can also be useful, but have not yet been applied to our problem.

3 Query Learning of Regular Languages

A regular language can be defined by the set of strings that are accepted by a deterministic finite automaton (DFA), defined as follows.

Definition 1. *A DFA defined over the alphabet Σ is a tuple $D = (Q, \Sigma, q_0, F, \delta)$, where:*

Q is a finite set of states;
$q_0 \in Q$ is the initial state;
$F \subset Q$ is the set of final or accepting states;
$\delta : Q \times \Sigma \to Q$ is the transition function.

A string is accepted by a DFA if there exists a sequence of transitions, matching the symbols in the string, starting from the initial state and ending in an accepting state.

The task of learning a regular language can be seen as learning a DFA that accepts the strings belonging to the language.

Query learning is concerned with the problem of learning an unknown concept using the answers provided by a teacher or oracle. In this context, the concept will be a DFA. There are several types of queries, but here we will focus on those sufficient to learn efficiently regular languages [14], namely:

membership queries the learner presents an instance for classification as to whether or not it belongs to the unknown language;

equivalence queries the learner presents a possible concept and the teacher either acknowledges that it is equivalent to the unknown language or returns an instance that distinguishes both.

Angluin presented an efficient algorithm [3] to identify regular languages from queries, the L* algorithm. This algorithm derives the minimum canonical DFA that is consistent with the answered queries.

3.1 The L* Algorithm

In this section we briefly describe the L* algorithm, in order to be able to present the proposed changes.

The L* algorithm defines a learner, in the query learning setting, for regular languages. This learner infers DFAs from the answers to membership and equivalence queries posed to a teacher. A teacher that can answer these types of queries is referred to as a minimally adequate teacher.

The instances used in membership queries are strings defined over the alphabet Σ. The concepts in equivalence queries are DFAs defined over that alphabet.

The information obtained from membership queries is used to define a function[1] $T : ((S \cup S \cdot \Sigma) \cdot E) \rightarrow \{0, 1\}$, where S is a nonempty finite prefix-closed set[2] of strings and E is a nonempty finite suffix-closed set of strings. The set $((S \cup S \cdot \Sigma) \cdot E)$ is the set of strings for which membership queries have been asked.

The function T has a value of 1 when the answer is positive, that is, when the string belongs to the language of the target DFA, and a value of 0 otherwise. It can be viewed as an observation table, where the rows are labeled by the elements of $(S \cup S \cdot \Sigma)$ and columns are labeled by the elements of E. For example, in Table 1, $S = \{\lambda\}$, $E = \{\lambda\}$ and $S \cdot \Sigma = \{0, 1, 2\}$.

Table 1. Initial L* table at the first conjecture

		λ	} E
S {	λ	0	
$S \cdot \Sigma$ [	(2)	0	
	(1)	0	
	(0)	0	

To represent a valid complete DFA, the observation table must meet two properties: closure and consistency. The observation table is *closed* iff, for each t in $S \cdot \Sigma$ there exists an $s \in S$ such that $row(t) = row(s)$. The observation table is *consistent* iff for every s_1 and s_2, elements of S, such that $row(s_1) = row(s_2)$ and for all $a \in \Sigma$, it holds that $row(s_1 \cdot a) = row(s_2 \cdot a)$.

The DFA $D = (Q, q_0, F, \delta)$ that corresponds to a *closed* and *consistent* observation table is defined by:

$Q = \{row(s) : s \in S\}$
$q_0 = row(\lambda)$
$F = \{row(s) : s \in S \wedge T(s) = 1\}$
$\delta(row(s), a) = row(s \cdot a)$

[1] Set concatenation $A \cdot B = \{ab | a \in A, b \in B\}$.

[2] A prefix-closed (suffix-closed) set is a set such that the prefix (suffix) of every set element is also a member of the set.

For example, Fig. 1 shows the DFA corresponding to the observation table shown in Table 2. Note that when a string belongs to both S and $S \cdot \Sigma$ then it is only represented once in the observation table (at the top part). This can be seen in Table 2. To obtain the transition function value for the initial state and symbol $0 \in \Sigma$, i.e. $\delta(row(\lambda), 0)$, one must lookup $row(\lambda \cdot 0)$, namely, the line labeled by (0). This line is shown in the top part of Table 2 because $(0) \in S$.

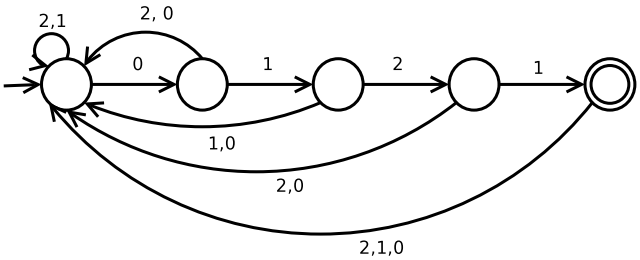

Fig. 1. Intermediate DFA

Table 2. L* table at the second conjecture

	λ	(1)	(2 1)	(1 2 1)
(0 1 2 1)	1	0	0	0
(0 1 2)	0	1	0	0
(0 1)	0	0	1	0
(0)	0	0	0	1
λ	0	0	0	0
(0 1 2 1 2)	0	0	0	0
(0 1 2 1 1)	0	0	0	0
(0 1 2 2)	0	0	0	0
(0 1 2 1 0)	0	0	0	0
(0 1 2 0)	0	0	0	0
(0 1 1)	0	0	0	0
(0 2)	0	0	0	0
(0 1 0)	0	0	0	0
(2)	0	0	0	0
(1)	0	0	0	0
(0 0)	0	0	0	0
(0 2 2 2)	0	0	0	0

It can be proved that any DFA consistent with the observation table, but different by more than an isomorphism, must have more states.

Starting with the initialization of the observation table, the L* algorithm proceeds to build a DFA. The DFA is presented to the teacher as an equivalence query.

Before a DFA can be generated from the observation table, it must verify two properties: the table must be *consistent* and *closed*. Therefore, a loop must be executed until the properties are met, updating the observation table along the way.

If the observation table is not *consistent*, then there exist $s_1, s_2 \in S$, $a \in \Sigma$ and $e \in E$ such that $row(s_1) = row(s_2)$ and $T(s_1 \cdot a \cdot e) \neq T(s_2 \cdot a \cdot e)$. The set E is augmented with $a \cdot e$ and the observation table is extended using membership queries.

If the observation table is not *closed*, then there exist an $s_1 \in S$ and an $a \in \Sigma$ such that $row(s_1 \cdot a)$ is different from all $row(s)$ with $s \in S$. The set S is extended with $s_1 \cdot a$ and the observation table is extended using membership queries.

Once the inner loop terminates, the DFA that corresponds to the observation table is presented to the teacher. If the teacher accepts this DFA the algorithm terminates. If the teacher returns a counter-example, then the counter-example and all of its prefixes are added to S, the table is extended using membership queries, and the algorithm continues with the first step (the loop that verifies the observation table properties).

For example, to learn the language $L = \{(0\ 1\ 2\ 1), (0\ 2\ 2\ 2)\}$, the algorithm starts a serie of queries (shown in Table 3) until it reaches a closed and consistent observation table, Table 1. It then poses the first conjecture. The first conjecture is a DFA with only one state, that accepts no strings. After receiving the string (0 1 2 1) as (a negative) reply to the first equivalence conjecture, the algorithm poses a long sequence of membership queries (shown in Table 3) until it finally reaches a point where the observation table is again both *closed* and *consistent*, as shown in Table 2. Then the second conjecture, shown in Fig. 1, is presented to the teacher. The process could be continued until the DFA shown in Fig. 2 is reached[3] that accepts only the strings present in the target language L.

4 Learning DFAs Using a More Powerful Teacher

Query learning algorithms, such as L*, described in the previous section, produce a very large number of queries. This makes their use with human teachers impractical.

Depending on the target language, the number of queries of each type varies. Nevertheless, the number of membership queries is typically the dominant part in the total query count.

Membership queries have two possible answers, positive or negative. The negative answer is used to restrict the target language (with respect to the universal language containing all strings). As such, it is reasonable to expect that

[3] The resulting DFA would have an additional non-accepting state, here omitted for clarity, where all the transitions that are not shown in the figure would converge.

Table 3. L* execution trace

L*			L* with filter
λ ? N	(0 1 2 1 2) ? N	(0 2 2 1) ? N	λ ? N
(0) ? N	(0 1 2 1 2 1) ? N	(0 1 0 2 1) ? N	(0) ? (e (0))
(1) ? N	(0 1 2 1 1 1) ? N	(2 2 1) ? N	(1) ? (s (1))
(2) ? N	(0 1 2 2 1) ? N	(1 2 1) ? N	(2) ? (s (2))
	(0 1 2 1 0 1) ? N	(0 0 2 1) ? N	
(conjecture)	(0 1 2 0 1) ? N	(0 1 2 1 2 1 2 1) ? N	(conjecture)
	(0 1 1 1) ? N	(0 1 2 1 1 1 2 1) ? N	
(0 0) ? N	(0 2 1) ? N	(0 1 2 2 1 2 1) ? N	(0 1) ? (e (0 1))
(0 1) ? N	(0 1 0 1) ? N	(0 1 2 1 0 1 2 1) ? N	(0 2) ? (e (0 2))
(0 2) ? N	(2 1) ? N	(0 1 2 0 1 2 1) ? N	(0 1 1) ? (p ((1 1)))
(0 1 0) ? N	(1 1) ? N	(0 1 1 1 2 1) ? N	(0 1 2) ? (e (0 1 2))
(0 1 1) ? N	(0 0 1) ? N	(0 2 1 2 1) ? N	(0 1 2 1) ? Y
(0 1 2) ? N	(0 1 2 1 2 2 1) ? N	(0 1 0 1 2 1) ? N	(0 1 2 2) ? (p ((1 2) (2)))
(0 1 2 0) ? N	(0 1 2 1 1 2 1) ? N	(2 1 2 1) ? N	(0 2 1) ? (p ((0 2) (1)))
(0 1 2 1) ? Y	(0 1 2 2 2 1) ? N	(1 1 2 1) ? N	(0 1 0 1 2 1) ? (p ((0) (0)))
(0 1 2 2) ? N	(0 1 2 1 0 2 1) ? N	(0 0 1 2 1) ? N	
(0 1 2 1 0) ? N	(0 1 2 0 2 1) ? N		(conjecture)
(0 1 2 1 1) ? N	(0 1 1 2 1) ? N	(conjecture)	

most languages will have a much larger count of negative membership queries than positive membership queries.

To deal with the large number of membership queries, that typically happens when learning non-trivial automata, we propose, in this work, to use a more powerful teacher. Should the answer to a membership query be negative, the teacher is requested to return additional information, namely, to identify a set of strings that would result also in negative answers to membership queries.

We consider three forms for the answer:

1. A string prefix – This form identifies the set of strings that start with the same prefix and that are also negative examples. Its use can be seen in Table 3, with the form *(s <string>)*.
2. A string suffix – The second form does the same as the first one, but with the string suffix. It identifies strings that end in the same manner and that are also negative examples. Its use can also be seen in Table 3, with the form *(e <string>)*.
3. A list of substrings – The third form can be used to specify a broader family of strings that are negative examples. Here one can identify strings by listing substrings that, when they are all present in a given string, in the same order, imply that the string is part of the described set and a negative example. For example, to identify the set of strings that contain two zeros, the reply would be the following list ((0)(0)), where (0) is a string with just one symbol, 0. Its use is also illustrated in Table 3, with the form *(p (<string1> ... <stringN>))*.

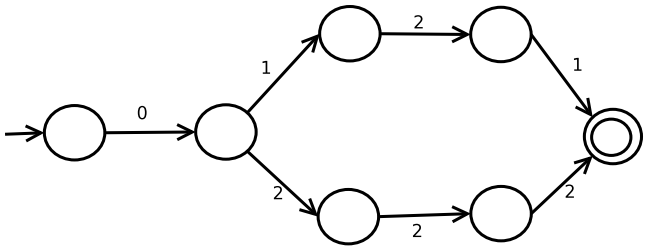

Fig. 2. Example DFA (extra state removed)

Note that these specifications can be viewed as non-deterministic finite automata (NFA).

Using the additional information, the learner can now find out the answer to a number of membership queries without making an explicit query, simply by matching the strings with the stored information using the NFA corresponding to the new answer form. This can clearly be seen in Table 3, where the same DFA is inferred with and without the proposed extension, resulting in an important reduction in the number of queries.

Although we are requiring more sophisticated answers from the teacher, this is a reasonable request when dealing with human teachers. A human teacher must have some informal definition of the target language in his mind, to use a query learning approach, and it is reasonable to expect that most negative answers could be justified using the proposed method. The use of a query learning method when an informal definition is already present in the human teacher's mind is necessary to obtain a minimal DFA with less effort than it would require to manually build one. As such, the extra required effort is a small one, since the human teacher would already have identified that justification in order to answer the original membership query. Moreover, in a graphical environment this could be easily implemented by allowing for the selection of parts of the query string using a mouse pointer (allowing for multiple-selection to indicate a list of substrings answer).

The proposed solution uses the L* algorithm as a "black box". A filter is placed between the teacher and the learner, which records the additional information returned by the teacher on negative membership query answers. This information is then used to reply, whenever possible, to the learner without consulting the teacher.

4.1 Example Results and Equivalence Queries

Table 4 shows the results obtained for the example DFA from Fig. 2. In this example, the strings used to reply (negatively) to the equivalence queries were (0 1 2 1), (0 2 2 2) and (2 2 2 2). Table 5 shows the distribution of membership query answers by type and the number of answers made by the filter using the information of each type of query answer.

Even in this simple example, the number of membership queries is substantially reduced making the method usable by human teachers (L* with filter (A) in Table 4). Further results with a real application are shown in the next section.

Table 4. Query count results - simple example

	Membership Query Numbers		Equivalence Query Numbers
	Positive	Negative	
L*	2	185	4
L* with filter (A)	2	13	4
L* with filter (B)	2	13	3

(A) - Filter applied to membership queries;
(B) - Filter applied to membership and equivalence queries.

Table 5. Query counts by type - simple example

	Start with	End with	Has parts	Unjustified	Total
Teacher answers	2	4	5	2	13
Filter use counts	72	21	79	-	172

The extra information returned by the teacher could also be used to answer some equivalence queries, namely those containing strings that are not part of the target language and can be detected by the NFA already recorded. For example, the DFA in Fig. 1 admits strings, containing two or more 0s, that do not belong to the language. This could be detected as it was already stated in the end of Table 3 by "(p ((0) (0)))".

However, to detect these cases it would be necessary to obtain the product automaton between the recorded NFA and the DFA proposed by the learner, a process with quadratic complexity on the number of states. Note also that the number of states not only increases with the complexity of the language, but also with the number of extended answers (answers with the new proposed forms). This is a costly operation and would only remove, in general, a small fraction of the equivalence queries (L* with filter (B) in Table 4).

5 Application to the Inference of Document Structure

To demonstrate the use of the proposed solution, we applied it to the inference of a grammar that describes the structure of technical articles. This work is part

of an ongoing effort to automatically derive an ontology describing the structure of technical articles.

The first step was described in [15] and resulted in the segmentation of source articles into a set of symbols. These symbols are: ConferenceTitle (0), Title (1), Author (2), AbstractTitle (3), AbstractText (4), IndexTitle (6), Index (7), SectionTitle (8), SubSectionTitle (10), SubSubSectionTitle (11), SimpleText (5), FormatedText (9), FigureCaption (12).

The next step in this effort is the inference of a DFA for technical articles, using the acquired symbols. This will later enable the inference of the ontology.

To apply the approach described in this paper, we assumed that:

- The ConferenceTitle is optional;
- There can be one or more Authors;
- The Index and IndexTitle are optional;
- A section can contain some of the text elements (SimpleText, FormatedText, FigureCaption) and lower level sections.

The equivalence queries were answered using the strings in Table 6.

Table 6. Strings used in equivalence queries

Q	Strings supplied to the algorithm
1	Title Author AbstractTitle AbstractText SectionTitle SimpleText
2	Title Title Author AbstractTitle AbstractText SectionTitle SimpleText
3	Title Author AbstractTitle AbstractText IndexTitle Index SectionTitle SimpleText
4	ConferenceTitle ConferenceTitle Title Author AbstractTitle AbstractText SectionTitle SimpleText
5	Title Author AbstractTitle AbstractText SectionTitle SimpleText SubSectionTitle SimpleText SubSubSectionTitle SimpleText

Table 7 shows the number of queries resulting from the use of the L* algorithm and of the proposed solution. The resulting DFA is shown in Fig. 3. Table 8 shows the distribution of membership query answers by type and the number of answers made by the filter using the information of each type of query answer.

As the results show, the number of negative membership queries is substantially reduced. Also, at least in this example, the negative membership queries are the largest in number. This is the case for automata that have few terminal states, relatively to the total number of states, a situation that is common in real cases.

As mentioned in Sect. 4.1, the information can be used to reduce the amount of equivalence queries (L* filter (B) in Table 7), but results in only a small reduction in the number of queries.

Table 7. Query count results

	Membership Query Numbers		Equivalence Query
	Positive	Negative	Numbers
L*	99	4118	6
L* with filter (A)	99	110	6
L* with filter (B)	99	110	5

(A) - Filter applied to membership queries;
(B) - Filter applied to membership and equivalence queries.

Table 8. Query counts by type

	Start with	End with	Has parts	Unjustified	Total
Teacher answers	11	10	88	1	110
Filter use counts	101	280	3627	-	4008

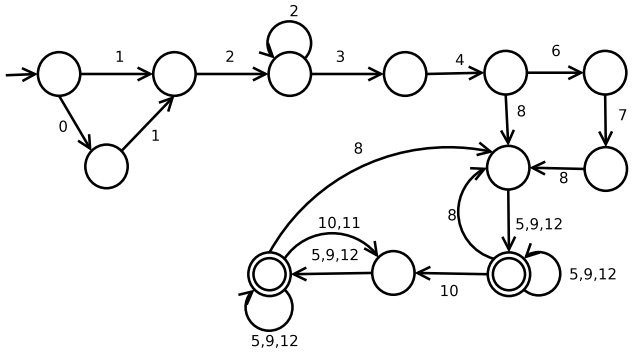

Fig. 3. DFA representing the article structure (extra state removed)

6 Conclusion

Query learning algorithms, when used with a human teacher, suffer from the excessive number of queries that are required to learn the target concept. In this work we have presented a simple extension to the well known L* algorithm that reduces this burden considerably.

The teacher is required to provide a list of sub-strings that, when present, implies that the query string does not belong to the target language. Using this extra information, many of the subsequent queries can be answered automatically by the filter. The additional information provided represents only a small amount of selection work by the user, greatly compensating for the reduction on the number of queries.

The solution is independent of the regular language query learning algorithm used, as long as the later relies mainly on membership queries. With it's use, such algorithms become a practical possibility in dealing with human teachers.

Acknowledgements

This work was partially supported by "Fundação para a Ciência e a Tecnologia" under research project PSOC/EIA/58210/2004 (OntoSeaWeb-Ontology Engineering for the Semantic Web).

References

1. Gold, E.M.: Complexity of automaton identification from given data. Inform. Control **37** (1978) 302–320
2. Pitt, L., Warmuth, M.: The minimum consistent DFA problem cannot be approximated within any polynomial. J. ACM **40** (1993) 95–142
3. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75** (1987) 86–106
4. Gold, E.M.: System identification via state characterization. Automatica **8** (1972) 621–636
5. Schapire, R.E.: The Design and Analysis of Efficient Learning Algorithms. MIT Press, Cambridge, MA (1992)
6. Nevill-Manning, C., Witten, I.H., Maulsby, D.L.: Modeling sequences using grammars and automata. In: Proceedings Canadian Machine Learning Workshop. (1994) 15–18
7. Hsu, C.N., Dung, M.T.: Generating finite-state transducers for semi-structured data extraction from the web. Information Systems **23** (1998) 521–538
8. Witten, I.H.: Adaptive text mining: inferring structure from sequences. Journal of Discrete Algorithms **2** (2004) 137–159
9. Laender, A.H.F., Ribeiro-Neto, B.A., da Silva, A.S., Teixeira, J.S.: A brief survey of web data extraction tools. SIGMOD Rec. **31** (2002) 84–93
10. Ribeiro-Neto, B.A., Laender, A.H.F., da Silva, A.S.: Extracting semi-structured data through examples. In: Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management, Kansas City, Missouri, USA, ACM (1999) 94–101
11. Adelberg, B.: NoDoSE - a tool for semi-automatically extracting semi-structured data from text documents. In: Proceedings ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA. (1998) 283–294
12. Califf, M.E., Mooney, R.J.: 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, Orlando, Florida, USA. (1999) 328–334
13. Soderland, S.: Learning information extraction rules for semi-structured and free text. Machine Learning **34** (1999) 233–272
14. Angluin, D.: Queries and concept learning. Machine Learning **2** (1988) 319–342
15. Martins, A.L., Pinto, H.S., Oliveira, A.L.: Towards automatic learning of a structure ontology for technical articles. In: Semantic Web Workshop at SIGIR 2004. (2004)