

# Rule Learning for Feature Values Extraction from HTML Product Information Sheets <sup>\*</sup>

Costin Bădică<sup>1</sup> and Amelia Bădică<sup>2</sup>

<sup>1</sup> University of Craiova, Software Engineering Department  
Bvd.Decebal 107, Craiova, 200440, Romania  
[c.badica@hotmail.com](mailto:c.badica@hotmail.com)

<sup>2</sup> University of Craiova, Business Information Systems Department  
A.I.Cuza 13, Craiova, RO-1100, Romania  
[ameliabd@yahoo.com](mailto:ameliabd@yahoo.com)

**Abstract.** The Web is now a huge information repository with a rich semantic structure that, however, is primarily addressed to human understanding rather than automated processing by a computer. The problem of collecting product information from the Web and organizing it in an appropriate way for automated machine processing is a primary task of software shopping agents and has received a lot of attention during the last years. In this paper we assume that product information is represented as a set of feature-value pairs contained in an HTML product information sheet that is usually formatted using HTML tables. The paper presents a technique for learning extraction rules of product information from such product information sheets. The technique exploits the fact that the Web pages that represent product information of a certain producer are generated on the fly from the producer database and therefore they exhibit uniform structures. Consequently, while the extraction task is executed manually for a few information items by a human user, a general-purpose inductive learner (we have used FOIL in our experiments) can learn extraction rules that will be further applied to the current and other product information sheets to automatically extract other items. The input to the learning algorithm is a relational description of the HTML document tree that defines the HTML tree nodes types and the relationships between them. The approach is demonstrated with appropriate examples, experimental results, and software tools.

## 1 Introduction

The Web is now a huge information repository that is characterized by i) *high diversity*, i.e. the Web information covers almost any application area, ii) *disparity*, i.e. the Web information comes in many formats ranging from plain and structured text to multimedia documents and iii) *rapid growth*, i.e. old information is continuously being updated in form and content and new information is

---

<sup>\*</sup> The research described here was partly supported with funding from Syncro Soft  
<http://www.oxygenxml.com>

constantly being produced. The HTML markup language is the *lingua franca* for publishing information on the Web. In our opinion these facts explain the significant growth of the interest in the field of automatizing the tasks of information extraction (IE hereafter) from HTML information sources.

IE was developed as part of DARPA's MUC program and originally was concerned with locating specific pieces of information in text documents written in natural language ([8]). The field then expanded rapidly to cover extraction tasks from networked documents including HTML and XML.

Paper [10] identified two major types of IE: i) IE from unstructured texts that treat the document as a string of characters and ii) IE from (semi-)structured texts that treat the document as a tree.

Traditional IE from strings uses techniques from computational linguistics based on finite-state methodologies. Paper [3] cites two approaches: i) the *local approach* concerned with enhancing document parsers with a set of extraction patterns and ii) the *global approach* concerned with applying grammatical inference methods to identify the language of the target document.

IE from tree documents exploits the structural information conveyed by the document tree. An important advantage of this approach is that the extracted item can depend on its structural context in a document, while this information is lost in the event the tree document is treated linearly as a string ([6]). Two recent IE approaches in this class are: i) *tree automata induction* ([6]) and ii) *learning subtree delimiters* ([3]).

This paper presents a methodology and experimental results on the application of first order rule learning to extract information from HTML documents structured as trees. The experiments were carried out in the area of collecting product information from the Web, an important task in product data integration ([4, 1]). However, the approach is general enough to be useful in other application areas that require IE from tree structured documents including XML and XHTML, as for example in wrapper development for querying bibliographic databases in digital libraries.

The rest of the paper is structured as follows. Section 2 contains an overview of the stages in the IE process. In section 3 we present the relational representation of HTML document trees. In section 4 we show how the task of learning IE rules from a tree document can be mapped to FOIL ([12]). Section 5 contains a brief description of the software tools we have used in our experiments. Section 6 presents the experiments performed and the results obtained in terms of precision and recall measures. Section 7 cites related work highlighting similarities with and differences from our work. Section 8 contains some conclusions and points to future research.

## 2 Overview of the Extraction Process

The IE process was structured as a linear sequence of stages:

- i) *Crawl or browse the Web and download HTML pages of interest.* This step is quite straightforward. Either a human user is browsing the Web to download

- interesting pages or the task of Web navigation and page download is automatized by means of a crawler component ([2]). The result of this step is a collection of pages that were fetched and stored locally on the user machine.
- ii) *Preprocess and convert the document from HTML to XHTML.* In this step the input HTML document is cleaned and converted to a well-formed XML document written in XHTML and structured as a tree.
  - iii) *Manually extract a few examples.* In this step the user loads a few XHTML pages from the local repository and performs a few extraction tasks. The result is a set of annotated XHTML documents with the extracted fields.
  - iv) *Parse the annotated XHTML documents and generate a relational representation as input for the learner.* In this step the annotated XHTML documents are parsed and converted into input for the learning program. The result will contain the target relation, the sets of positive and negative examples and the background relations.
  - v) *Apply the learning algorithm and obtain the IE rules.* In this step the learning program is executed on the input generated in the previous step. The result is a set of first order IE rules.
  - vi) *Apply the rules to extract information.* In this step the user applies the learnt IE rules to new XHTML pages in order to automatically extract new information. In principle there are two possibilities to perform this task: i) the target document is converted to a relational representation in Prolog and the first order IE rules are applied to the Prolog representation or ii) the first order IE rules are mapped to XSLT and the resulted XSLT is applied to the target XHTML document.

### 3 Relational Representation of Document Trees

An XHTML document is composed of a structural component and a content component.

The structural component consists of the set of document nodes or elements. The document elements are nested into a tree like structure. Each document element has assigned a specific tag from a given finite set of tags  $\Sigma$ . There is a special tag  $text \in \Sigma$  that designates a text element.

The content component of a document consists of the actual text in the text elements and the attribute-value pairs attached to the other document elements.

The structure of XHTML documents can be represented as unranked ordered trees, also known as  $\Sigma$ -trees. According to [11], the set  $\mathcal{T}_\Sigma$  of  $\Sigma$ -trees is defined inductively as follows:

- i) if  $\sigma \in \Sigma$  then  $\sigma \in \mathcal{T}_\Sigma$ ;
- ii) if  $\sigma \in \Sigma$  and  $t_1, \dots, t_n \in \mathcal{T}_\Sigma$ ,  $n \geq 1$ , then  $\sigma(t_1, \dots, t_n) \in \mathcal{T}_\Sigma$ .

Note that there is no a priori bound on the number of children of a node in a  $\Sigma$ -tree, i.e. the tree is *unranked*, and note also that the set of children of a given node in a  $\Sigma$ -tree is ordered, i.e. the tree is *ordered*.

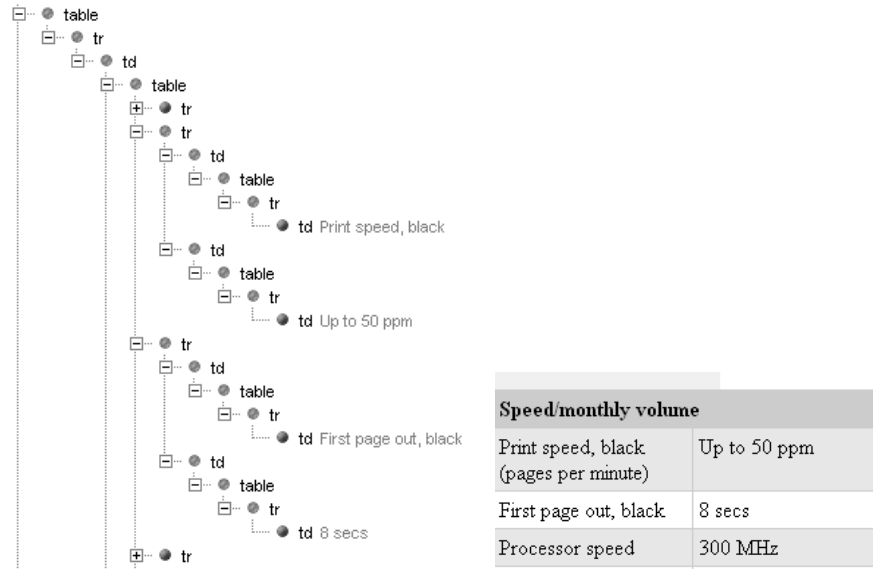


Fig. 1. An XHTML document fragment and its graphic view

*Example 1.* Consider the Hewlett Packard's site of electronic products<sup>1</sup> and the task of IE from a product information sheet for printers. The printer information is displayed in a two-column table as a set of feature-value pairs. Our task is to extract the names and/or the values of the printer features. This information is stored in the leaf elements of the page. Figure 1 displays in the left panel the XHTML tree of a fragment of this document and in the right panel the graphical view of this fragment as a two-column table. Figure 2 displays this XHTML document fragment as a  $\Sigma$ -tree. ■

Recall that our task in this section was to devise a relational representation for an XHTML document tree. We assign a unique identifier (an integer value) to each node of the tree. Let  $\mathcal{N}$  be the set of all node identifiers. This set can be partitioned into the set  $\mathcal{N}_N$  of non-text node identifiers and the set  $\mathcal{N}_T$  of text node identifiers. In what follows a node is identified by its unique identifier.

The structural component of an XHTML document tree can be represented using the following three relations:

- i)  $child \subseteq \mathcal{N}_N \times \mathcal{N}$  defined as  $(child(P, C) = true) \Leftrightarrow (P \text{ is the parent of } C)$ .
- ii)  $next \subseteq \mathcal{N} \times \mathcal{N}$  defined as  $(next(L, N) = true) \Leftrightarrow (L \text{ is the left sibling of } N)$ .
- iii)  $tag \subseteq \mathcal{N} \times \Sigma$  defined as  $(tag(N, T) = true) \Leftrightarrow (T \text{ is the tag of node } N)$ .

In order to be able to represent the content component of an XHTML document tree, we introduce two more sets: the set  $\mathcal{S}$  of content elements, which are basically the strings attached to text elements and the values assigned to element attributes, and the set  $\mathcal{A}$  of attributes. With these notations, the content

<sup>1</sup> <http://www.hp.com>

<i>html</i> ( ...	<i>tag</i> (0. <i>html</i> )		
<i>table</i> (	<i>tag</i> (100, <i>table</i> )		
<tr>(...),</tr>	<i>tag</i> (101, <i>tr</i> )	<i>child</i> (100,101)	
<tr(< td=""> <td><i>tag</i>(102,<i>tr</i>)</td> <td><i>child</i>(100,101)</td> </tr(<>	<i>tag</i> (102, <i>tr</i> )	<i>child</i> (100,101)	
<td><i>td</i>(<i>table</i>(<i>tr</i>(<i>td</i>(<i>text</i>))),</td> <td><i>tag</i>(103,<i>tr</i>)</td> <td><i>child</i>(100,102)</td>	<i>td</i> ( <i>table</i> ( <i>tr</i> ( <i>td</i> ( <i>text</i> ))),	<i>tag</i> (103, <i>tr</i> )	<i>child</i> (100,102)
<td><i>td</i>(<i>table</i>(<i>tr</i>(<i>td</i>(<i>text</i>))),</td> <td><i>tag</i>(107,<i>td</i>)</td> <td><i>child</i>(100,103)</td>	<i>td</i> ( <i>table</i> ( <i>tr</i> ( <i>td</i> ( <i>text</i> ))),	<i>tag</i> (107, <i>td</i> )	<i>child</i> (100,103)
),	<i>tag</i> (108, <i>table</i> )	<i>child</i> (101,107)	
),	<i>tag</i> (109, <i>tr</i> )	<i>child</i> (107,108)	
<tr(< td=""> <td><i>tag</i>(110,<i>td</i>)</td> <td><i>child</i>(108,109)</td> </tr(<>	<i>tag</i> (110, <i>td</i> )	<i>child</i> (108,109)	
<td><i>td</i>(<i>table</i>(<i>tr</i>(<i>td</i>(<i>text</i>))),</td> <td><i>tag</i>(111,<i>text</i>)</td> <td><i>child</i>(109,110)</td>	<i>td</i> ( <i>table</i> ( <i>tr</i> ( <i>td</i> ( <i>text</i> ))),	<i>tag</i> (111, <i>text</i> )	<i>child</i> (109,110)
<td><i>td</i>(<i>table</i>(<i>tr</i>(<i>td</i>(<i>text</i>))),</td> <td></td> <td><i>child</i>(110,111)</td>	<i>td</i> ( <i>table</i> ( <i>tr</i> ( <i>td</i> ( <i>text</i> ))),		<i>child</i> (110,111)
),			
<tr>(...),</tr>	<i>content</i> (111,	<i>next</i> (101,102)	
...	'Print speed, black' )	<i>next</i> (102,103)	
)		<i>next</i> (103,104)	
...	<i>attribute_value</i> (108,		
)	<i>border</i> , '0' )		

**Fig. 2.** The structure of the XHTML fragment from figure 1 represented as a  $\Sigma$ -tree and a part of its relational representation

component of an XHTML document tree can be represented using the following two relations:

- i)  $content \subseteq \mathcal{N}_T \times \mathcal{S}$  defined as  $(content(N, S) = true) \Leftrightarrow (S \text{ is the string contained by the text node } N)$ .
- ii)  $attribute\_value \subseteq \mathcal{N}_N \times \mathcal{A} \times \mathcal{S}$  defined as  $(attribute\_value(N, A, S) = true) \Leftrightarrow (S \text{ is the text value of the attribute } A \text{ of the node } N)$ .

*Example 2.* The right panel of figure 2 displays a part of the relational representation of the XHTML document tree shown on the left panel. ■

The relational representation of document trees introduced in this section makes convenient the definition of first-order IE rules. An IE rule defines a relation  $extract \subseteq \mathcal{N}$  such that  $(extract(N) = true) \Leftrightarrow (N \text{ is an extracted node})$ . Usually, the extraction process is focused on text elements, so we can safely assume that  $extract \subseteq \mathcal{N}_T$ .

The definition of an IE rule consists of a set of clauses. The head of a clause is  $extract(N)$  and the body of a clause is a conjunction of positive and negative literals made of the relations  $child$ ,  $next$ , and  $tag$  ( $attribute\_value$  could be used as well to additionally check node attributes). The relation  $content$  can be used to find the actual content of the extracted nodes.

*Example 3.* Assuming that we want to extract all the text nodes of the XHTML document from figure 1 that have a grand-grand-parent of type *table* that has a parent that has a right sibling, we can use the following IE rule:

$$\begin{aligned}
 extract(A) \leftarrow & \\
 & tag(A, text) \wedge child(B, A) \wedge child(C, B) \wedge child(D, C) \wedge tag(D, table) \wedge \\
 & child(E, D) \wedge next(E, F)
 \end{aligned}$$

■

Obviously, the manual writing of IE rules of this type is a slow and difficult process requiring a careful analysis of the structure of the target XHTML document. Fortunately, it is possible to learn first order IE rules with a general purpose relational learning program and this issue is the focus of the next section of the paper.

## 4 Using FOIL to Learn Extraction Rules

FOIL – First Order Inductive Learner is a general purpose first order (also known as relational) inductive learning program developed by John Ross Quinlan and his team at the beginning of the '90s ([12]). Even if it is not in the scope of the paper to give a detailed description of how FOIL works, we decided to give a brief overview of first order inductive learning in order to make the paper self contained. For more details see [12] and [9] (chapter 10).

In first order inductive learning the training data comprises the following two components:

- i) The *target relation*, i.e. the relation that is learnt. It is defined extensionally as a set of tuples. Usually the tuples are partitioned into a set of positive tuples and a set of negative tuples.
- ii) The *background relations*, usually defined extensionally as sets of tuples.

The goal of first order inductive learning is to construct a logic program that represents an appropriate intensional definition of the target relation in terms of the background relations and, optionally itself, if recursive definitions are allowed. The learnt definition must cover all the positive tuples (or a large fraction of them) and no negative tuples (or a small fraction of them) from its extensional definition.

FOIL requires as input the information about the target and the background relations, as mentioned above, and the definition of the relations arguments types, as sets of constants. FOIL's output is a set of clauses that represents a disjunctive definition of the target relation. The set of clauses is constructed iteratively, one clause at a time, removing all the covered positive tuples until an empty set remains. Every clause is constructed using an heuristic hill climbing search strategy that is guided by the information gain, determinate literals and the clause complexity (see [12] for details).

An important step in our IE experiments was to prepare the input for FOIL. In what follows we assume that during the training stage the user selects a single XHTML document – the training document, and performs a few extraction tasks on it. The problem is to map the training data into input for FOIL.

First, we must define the relations arguments types. We used a single type that represents the set  $\mathcal{N}$  of all the nodes found in the training document.

Second, we must define the target relation *extract*. We used only the nodes extracted during the training stage from the training document, as the set of positive examples. FOIL uses the closed world assumption by considering all the other nodes in the training document as negative examples.

Third, we must define the background relations *next* and *tag*. For convenience, we have replaced the relation *tag* with a set of relations  $r_\sigma$ , for all  $\sigma \in \Sigma$ , defined as  $r_\sigma = \{N | \text{tag}(N, \sigma) = \text{true}\}$ .

*Example 4.* We consider a FOIL input example taken from our experiments. The input is split into the following sections: argument types ( $N$ ), target relation (*extract*, only positive examples), and background relations (*child*, *next*, *table*, ..., *span*).

#N: 0,1,2,3,4, ..., 1096,1097.

```

extract(N)
1004
1006
...
1056
1058
.
*child(N,N)
0,1
0,2
1,3
1,4
...
1094,1097
.
*next(N,N)
1,2
3,4
...
1091,1092
.
*table(N)
26
27
...
991
.
...
*span(N)
250
251
...
980
.

```

■

Additionally, FOIL can be parameterized from the command line. All the parameters have default values, but we found useful to control explicitly the following options:

- i) The use or not of negative literals in the body of a clause.
- ii) The maximum variable depth in literals. Variables occurring in the clause head have depth 0 and a new variable in the literal has depth one greater than the maximum depth of its existing variables.
- iii) And finally, to set the minimum clause accuracy to an appropriate value. The clause accuracy represents the percentage of the positive tuples from the set of all tuples covered by the relation. FOIL will not accept any clause with an accuracy lower than this value.

*Example 5.* This example shows a FOIL command-line used by our experiments.

```
foil -d8 -a40 -v3 <ex10.d >ex10.o
```

The parameter `d8` is used for setting the maximum variable depth to 8. The parameter `a40` is used for setting the minimum clause accuracy to 40 %. The parameter `v3` is used to set the level of verbosity of FOIL's output to 3 (minimum is 0 and maximum is 4), i.e. the output will include quite detailed information about the FOIL's search process. The parameters `ex10.d` and `ex10.o` are the input and output files. ■

## 5 Software Tools

We have developed a set of prototype software tools to support our experiments with rule learning for IE from HTML. The tool set implements the scenario outlined in section 2 with the restriction that the training stage is based on a single training document. The tool set incorporates the following on-the-shelf software components:

- i) The machine learning program FOIL. This component is called from the main program through a command line.
- ii) The Xerces<sup>2</sup> library for XML processing. This component is used to read an XHTML document into a DOM tree. The DOM tree is then mapped onto the format required as input for FOIL using a breadth-first traversal.
- iii) The Tidy<sup>3</sup> library for HTML document cleaning and pre-processing. This component is used in the pre-processing stage of HTML documents for converting them to XHTML. We have also developed a custom XSLT stylesheet to remove redundant HTML elements like *script* and *style*.

The development of a Java-based GUI shell for our set of tools is underway. It will support the user in performing the following operations: convert HTML

<sup>2</sup> <http://xml.apache.org/xerces2-j/index.html>

<sup>3</sup> <http://www.w3c.org/People/Ragget/tidy/>

files to XHTML, load an XHTML document for training, manually select with the mouse some examples for extraction, generate the input for FOIL from the training document and the manually extracted information, run FOIL, visualize the learnt IE rules, convert the IE rules to XSLT, visualize the XSLT IE rules, run the XSLT IE rules on a target XHTML document and visualize the output.

## 6 Experimental Results

We ran a series of experiments of rule learning for IE from the Hewlett Packard’s Web site. The task was to extract the printers feature values from their information sheets.

We selected an experimental set of 30 documents representing information sheets for HP printers. The printer information is represented in two column HTML tables (see figure 1). The feature values are stored in the second column. Each of the selected documents contains between 28 and 38 features.

We used training examples from a single document representing the information sheet of the HP Business Inkjet 2600 (C8109A) printer. This document contains 28 positive examples and a total of 1098 nodes. We performed learning experiments with 1, 2, 3, 4, 12, 24 and 28 positive examples selected from this set. The results of the learning stage were applied to all of the 30 documents, measuring the precision and recall indicators. These values were computed by averaging the precision and recall for each individual document.

The experiments were divided in two classes:

- i) In the first class we have ignored the *next* relation as a background relation of FOIL’s input.
- ii) In the second class we have added the *next* relation to the set of background relations of FOIL’s input.

In all the experiments we ignored the attribute information of HTML elements (i.e. the relation *attribute\_value* introduced in section 3).

For each class of experiments we have varied explicitly the following parameters: the use or not of negated literals in rule bodies (FOIL parameter), the minimum clause accuracy (FOIL parameter), and the number of positive examples. We also set the maximum variable depth to 8 and used the closed world assumption to let FOIL generate the set of negative examples in all the experiments.

The results of the experiments in the first and second class are summarized in tables 1 and 2 respectively.

*Example 6.* This example shows the rule learnt for our printer data case in experiment 16 from table 2.

$$\begin{aligned} \text{extract}(A) \leftarrow \\ & \text{child}(B, A) \wedge \text{tag}(A, \text{text}) \wedge \text{tag}(B, \text{td}) \wedge \neg \text{next}(A, \_1) \wedge \neg \text{next}(\_1, A) \wedge \\ & \text{child}(C, B) \wedge \text{child}(D, C) \wedge \neg \text{next}(\_1, C) \wedge \text{child}(E, D) \wedge \text{child}(F, E) \wedge \\ & \neg \text{next}(F, \_1) \wedge \neg \text{next}(E, \_1) \end{aligned}$$

■

**Table 1.** Experiments without the use of the background relation *next*

Exper. no.	Neg. lit.	No. of pos. examples	Clause acc.	Precision	Recall
1	No	12	10 %	0.334	0.980
2	Yes	12	0 %	0.355	1
3	No	24	20 %	0.334	0.980
4	Yes	24	0 %	0.409	0.970
5	No	28	20 %, 30 %	0.334	0.980
6	Yes	28	0 %, 20 %	0.409	0.970

**Table 2.** Experiments with the use of the background relation *next*

Exper. no.	Neg. lit.	No. of pos. examples	Clause acc.	Precision	Recall
1	No	1	0 %	0.332	1
2	Yes	1	0 %	0.411	1
3	No	2	0 %	0.730	0.800
4	Yes	2	0 %	0.440	1
5	No	3	0 %, 5 %	0.698	1
6	Yes	3	0 %	0.440	1
7	No	4	0 %	0.332	1
8	No	4	10 %	0.730	0.800
9	Yes	4	0 %	0.448	1
10	No	12	20 %, 40 %	0.855	0.800
11	Yes	12	20 %	1	0.800
12	No	24	20 %	0.698	1
13	No	24	60 %	0.855	0.800
14	Yes	24	80 % (default)	1	0.800
15	No	28	80 % (default)	0.855	0.800
16	Yes	28	80 % (default)	1	1

Note that if  $X$  is an unbound variable and  $Y$  is a bound variable then  $\neg next(X, Y) = true$  means that  $Y$  is instantiated with the first child node of its parent node. Therefore we can introduce the relation *first* defined as  $(first(X) = true) \Leftrightarrow (X \text{ is the first child node of its parent node})$  and replace  $\neg next(X, Y)$  with  $first(Y)$ . Similarly, if  $X$  is a bound variable and  $Y$  is an unbound variable then  $\neg next(X, Y) = true$  means that  $Y$  is instantiated with the last child node of its parent node. Therefore we can introduce the relation *last* defined as  $(last(X) = true) \Leftrightarrow (X \text{ is the last child node of its parent node})$  and replace  $\neg next(X, Y)$  with  $last(X)$ .

*Example 7.* The rule from example 6 can be rewritten without negations:

```
extract(A) ←
  child(B, A) ∧ tag(A, text) ∧ tag(B, td) ∧ last(A) ∧ first(A) ∧ child(C, B) ∧
  child(D, C) ∧ first(C) ∧ child(E, D) ∧ child(F, E) ∧ last(F) ∧ last(E)
```

Note that this rule can be easily mapped to the following XPath expression:

```
//*[*[position()=last()]/*[position()=last()]/*[position()=1]/
td/node()[last()=1]/self::text()
```

■

After a careful analysis of the results of our experiments, the following conclusions were drawn:

- i) In all the experiments with the precision lower than 0.500 the learnt IE rule also extracted the feature name of every extracted feature value. This explains the low values of the precision in these experiments.
- ii) The low precision values in the experiments of the first class clearly shows the importance of exploiting the information conveyed by the *next* relation for IE from tree structured documents.
- iii) The use of negated literals in rule bodies improves significantly the precision of the IE rules in the case of using the *next* relation. But note that in all the experiments the use of  $\neg$  could be replaced with predicates *last* and *first*.
- iv) The learnt IE rules were able to extract all the feature values (i.r. recall was 1) even when for training was used a single positive example. This indicates a minimum user effort to manually extract a single item from the training document.
- v) In the case when we used for training as positive examples all the fields that must be extracted from the training document and we allowed the presence of negated literals in rule bodies, the precision and recall were 1.

## 7 Related Work

With the rapid expansion of the Internet and the Web, the field of IE from HTML attracted a lot of researchers during the last decade. Clearly, it is impossible to mention all of their work here. However, at least two papers that are more closely related to the work reported in our paper, can be cited – [5] and [3].

Paper [5] describes the relational learner SRV that uses a FOIL-like algorithm for learning first order IE rules from a text document represented as a sequence of lexical tokens. The relations used in the rule bodies check various token features like: length, position in the text fragment, if they are numeric or capitalized, a.o. SRV has been adapted to learn IE rules from HTML. For this purpose new token features have been added to check the HTML context in which a token occurs. The most important similarity between SRV and our approach is the use of relational learning and a FOIL-like algorithm. The difference is that our approach has been explicitly devised to cope with tree structured documents, rather than string documents.

Paper [3] describes a generalization of the notion of string delimiters developed for IE from string documents ([7]) to subtree delimiters for IE from tree documents. The paper describes a special purpose learner that constructs a structure called candidate index based on trie data structures, which is very different from FOIL’s approach. Note however that the tree leaf delimiters described in that paper are quite similar to our IE rules. Moreover, the representation of reverse paths using the symbols  $Up(\uparrow)$ ,  $Left(\leftarrow)$  and  $Right(\rightarrow)$  can be easily simulated in our IE rules using the relations *child* and *next*.

## 8 Conclusions

This paper brings experimental evidence in support of the usefulness of first order rule induction for IE from HTML documents. As future research directions we would like to mention: i) the development of a GUI support for the IE approach outlined in this paper, in order to make it feasible for real users, ii) the investigation into the possibility of automatic translation of first order IE rules to XSLT or other XML query language and iii) experimenting with our approach in other application areas and for significantly larger collection of Web pages in order to assess its scalability and generality.

## References

1. Bădică, C., Bădică, A., Lițoiu, V.: Enhancing WWW E-Commerce by Acquiring and Managing Product Knowledge. In: *Proceedings of TAINN'03*, vol.E-7, Çanakkale, Turkey (2003) 684–692.
2. Chakrabarti, S.: *Mining the Web. Discovering Knowledge from Hypertext Data*. Morgan Kaufmann Publishers, (2003).
3. Chidlovskii, B.: Information Extraction from Tree Documents by Learning Subtree Delimiters. In: *Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, Acapulco, Mexico (2003), 3–8.
4. Fensel, D., Ding, Y., Omelayenko, B., Schulten, E., Botquin, G., Brown, M., Flet, A.: Product Data Integration. *IEEE Intelligent Systems*, Vol. 16, No.4, IEEE Computer Society (2001) 54–59.
5. Freitag, D.: Information extraction from HTML: application of a general machine learning approach. In: *Proceedings of AAAI'98*, (1998), 517–523.
6. Kosala, R., Bussche, J. van den, Bruynooghe, M., Blockeel, H.: Information Extraction in Structured Documents using Tree Automata Induction. In: *Principles of Data Mining and Knowledge Discovery, 6<sup>th</sup> European Conference*, Helsinki, Finland, LNAI 2431, (2002), 299–310.
7. Kushmerick, N.: Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, No.118, Elsevier Science (2000), 15–68.
8. Lenhart, W., Sundheim, B.: A Performance Evaluation of Text-Analysis Technologies. In: *AI Magazine*, 12(3) (1991), 81–94.
9. Mitchell, T.M.: *Machine Learning*, McGraw-Hill, (1997).
10. Muşlea, I.: Extraction Patterns for Information Extraction Tasks: A Survey. In: *AAAI-99 Workshop on Machine Learning for Information Extraction*, (1999).
11. Neven, F.: Automata Theory for XML Researchers. *SIGMOD Record*, 31(3) (2002), 39–46.
12. Quinlan, J. R., Cameron-Jones, R. M.: Induction of Logic Programs: FOIL and Related Systems, *New Generation Computing*, 13, (1995), 287–312.