From HTML Documents to Web Tables and Rules

Kai Simon Georg Lausen

Institut für Informatik, Universität Freiburg
Georges-Köhler-Allee, Gebäude 51
79110 Freiburg i.Br., Germany

{ksimon,lausen} AT informatik.uni-freiburg.de

Harold Boley

Institute for Information Technology – e-Business
National Research Council of Canada
Fredericton, NB, E3B 9W4, Canada

Harold.Boley AT nrc.gc.ca

ABSTRACT

We present a browser-extending Semantic Web extraction system that maps HTML documents to tables and, where possible, to rules. First, the basic data extractor ViPER distills and reorganizes semi-structured information into a tabular data structure, which can again be browsed and/or submitted to further machine processing. Second, exemplifying the latter, the extended knowledge extractor Rex ViPER mines the resulting tables for structural properties and functional dependencies. Rules are generated to obtain a more compact and manageable, often also enriched, knowledge representation. The resulting fully structured information, RuleML-serialized facts and rules, can be stored along with the orginal documents, queried by rule engines such as OO jDREW and FLORID, and interchanged between Web Services. Thus Rex ViPER contributes to automating the construction of a machine-processable Semantic Web.

Categories and Subject Descriptors

H.3.m [**Information Storage and Retrieval**]: Miscellaneous—*Data Extraction, Wrapper Generation, Web*

General Terms

Rule-based languages

Keywords

Data extraction, data record alignment,Rule-based languages

1. INTRODUCTION

The Internet has revolutionized the way we search for information, extract information, and also the way we exchange information. However, due to the fact that a huge amount of information available on the Web is only accessible through presentation-oriented HTML pages, most of the interactions take place between humans. In order to pave the way for the Semantic Web, which focuses on computer-computer interactions, we require techniques empowering machines to extract, represent, and process information resources, on behalf of humans. The benefits a user would obtain from these techniques are obvious. For instance, the system could extract (XML) representations from various Web sources, semantically integrate and process them, e.g. finding regularities in the form of (RuleML) rules, and present results in a suitable manner.

First, we start with the description of our fully automatic Web data extraction system, called ViPER [17], which has already been embedded into a wide-spread Web browser. Equipped with the plugin the user is able to extract information from arbitrary Web pages containing multiple similarly structured data records, which is a typical characteristic of static Web catalogs as well as dynamic Web pages. Due to the fact that these sites are often filled with information from back-end databases by predefined templates or server-side scripts, the extraction process can be seen as reverse engineering on the basis of materialized database views which have been published in HTML pages. Consequently, we reorganize the extracted information as an interactive XML-serialized table opening the door to a wide range of post-processing functionalities.

After the extraction process and reorganization of the insulated data records into a tabular representation, we mine structural properties and functional dependencies with an extension of the ViPER system referred to as the Rule-extracting ViPER system, Rex ViPER for short. Previous work in rule extraction includes the LISp-Miner project for discovering statistical association rules in databases[1] and the XRML framework for obtaining rules from natural language [11].

In this paper, we focus on data-driven methods to discover correlations inside the resulting table, both in their columns and rows. Rex ViPER employs dependency mining to generate rules which become serialized in XML, specifically in RuleML [4]. We will focus on rules that enable the (column and row) compactification and enrichment of tables. These rules can then derive the original and enriched information on demand. Such rules also benefit the table representation produced by the basic ViPER system because the remaining columns and rows become easier to navigate and administrate interactively. A balanced representation employing both tables and rules can result in improved information processing and management, and also benefits the

[1] [http://lispminer.vse.cz/]

ICEC'06, August 14–16, 2006, Fredericton, Canada.

user through a personalized view of each of the currently materialized HTML pages. To ensure the portability of Rex ViPER, two rule engines are employed to query representations consisting of tables (facts) and rules: the RuleML implementation OO jDREW [1] and the F-logic implementation FLORID [6].

2. WEB DATA EXTRACTION AND ALIGNMENT

Wrapper tools for extracting information from HTML pages started attracting major research interest during the midnineties. One significant characteristic is their degree of automation, reaching from specially designed standalone wrapper programming languages, for manual wrapper generation, over machine learning, and interactive approaches with more or less human interaction to fully-automatic wrapper tools. We refer the interested reader to [10] for a brief survey of different wrapping techniques. Whereas the majority of the approaches rely on user interaction to extract information, more recently, interest in automatically generated wrappers without human involvement has grown substantially.

2.1 Automatic Extraction

We developed a fully-automatic wrapper extraction tool named ViPER (**Vi**sual **P**erception-based **E**xtraction of **R**ecords). The original prototype controls the Gecko Engine of Mozilla over the XPCOM interface via the Java Framework JREX[2] which enables us to use the correction and rendering power of Gecko within Java. The principle assumption made is that a Web page contains at least two multiple spatially consecutive *data records* building a *data region* which exhibits some kind of structural and visible similarity. ViPER is then able to extract and discriminate the relevance of different repetitive information contents with respect to the user's *visual* perception of the Web page. Having identified the most relevant data region the tool extracts similar data records by declaring one data record contained in that data region as *pattern* and aligns matching data records utilizing a fast and robust *multiple sequence alignment* (MSA) technique. In [17] we already showed the accuracy of the extraction process by comparing our system with the results reported by existing fully-automatic state-of-the-art wrapping tools. Meanwhile, we have developed a slim plugin realization of ViPER for the Mozilla Firefox Web browser with additional interactive post-processing functionalities. Most of these functionalities pertain modifications of the resulting table such as labeling or sorting of the aligned columns. On the other side the user has also the possibility to manually select different patterns from a ranked list of reported patterns after the extraction process. Some of these patterns could for instance match structured advertisements or links pointing to following pages.

Figure 1 depicts our running example on the basis of which we like to illustrate both the extraction and rule mining process. More examples and their corresponding outputs can be found under the following link[3]. The Web page in 1A) contains similar structured information denoted as data records where each of them consists of the following *data items*: a linked picture of the product, a link labeled with product description, product code, color, different price information and optional shipping information. Inside the Web page the data records are arranged in a 3x4 dimensional table and therefore build a compact data region. The most relevant extraction pattern suggested by ViPER consists of 59 HTML tag elements whereas text content becomes abstracted by an artificial element symbolize by <TEXT>. Within these 59 elements only <A>,<IMG> and <TEXT> elements become displayed in the final table representation which finally sums up to 14 data items.

[2][http://jrex.mozdev.org/]

[3][http://dbis.informatik.uni-freiburg.de/ViPER/Rex/Rules.html]

2.2 Automatic Alignment

After the extraction process the system attempts to align the data records. In [17] we described our partial data alignment approach based on global sequence alignment techniques, tree structure information and string similarity. The reported benchmarking results performed on a labeled third party data set underpin the accuracy and efficiency of our data alignment technique. The advantage of aligned data records is that it is easy to store the data records in a database, export them as XML or synchronize them with data extracted from other Web pages. The result of the alignment process of our example Web site is shown in figure 1B). Each data record corresponds to a row in the table and aligned data items belong to the same column. We have to mention that in the majority of cases the number of data items in the pattern doesn't determine the final number of columns because of additional data items in data records matching the pattern with respect to the modified edit-distance computation as described in [17].

3. TABLE MINING

In this paper, we focus on data-driven methods to discover correlations between columns of the resulting aligned table. Hence, the system explores the rich dependency structure that often prevails in most real-world Web data based on the aligned data records. We apply heuristics and statistical analysis to identify both structural and functional properties. Our approach is totally data-driven and instance/sample-based because we can only identify properties from the presented Web view and have no access to schema information or catalog statistics. To ensure that the number of data records exceed a statistical significance we accumulate data records from so called following pages and insert them into our resulting table if available.

3.1 Preliminary Definitions

Before we describe the mining techniques we want to specify our table in more details. Let t define the resulting table after the alignment process and $K \times N$ its dimension where K denotes the total number of rows and N the total number of columns. A particular data item in the i-th row and the j-th column is addressed by $t.s_{i,j}$ or shortly $s_{i,j}$ with $1 \leq i \leq K, 1 \leq j \leq N$. The short notation is practical because in our case we always refer to the same relation t. A common notation to refer to a special entry contained in a row or a column is $R_i := s_{i,\cdot}$ or $C_j := s_{\cdot,j}$, respectively.

Next we define some statistical dimensions describing the table in more details. If we focus on a specific column C_j with $D_j \leq N$ *distinct values*, say $\mathcal{D}_j := \{1, 2, ..., D_j\}$, then $n_{k,j}$ denotes the *number of times* value $k \in \mathcal{D}_j$ occurs in the j-th column. With *total sum* $n_j := \sum_{k=1}^{D_j} n_{k,j}$ where the *relative frequency* of value k corresponds to $p_{k,j} := n_{k,j}/n_j$,

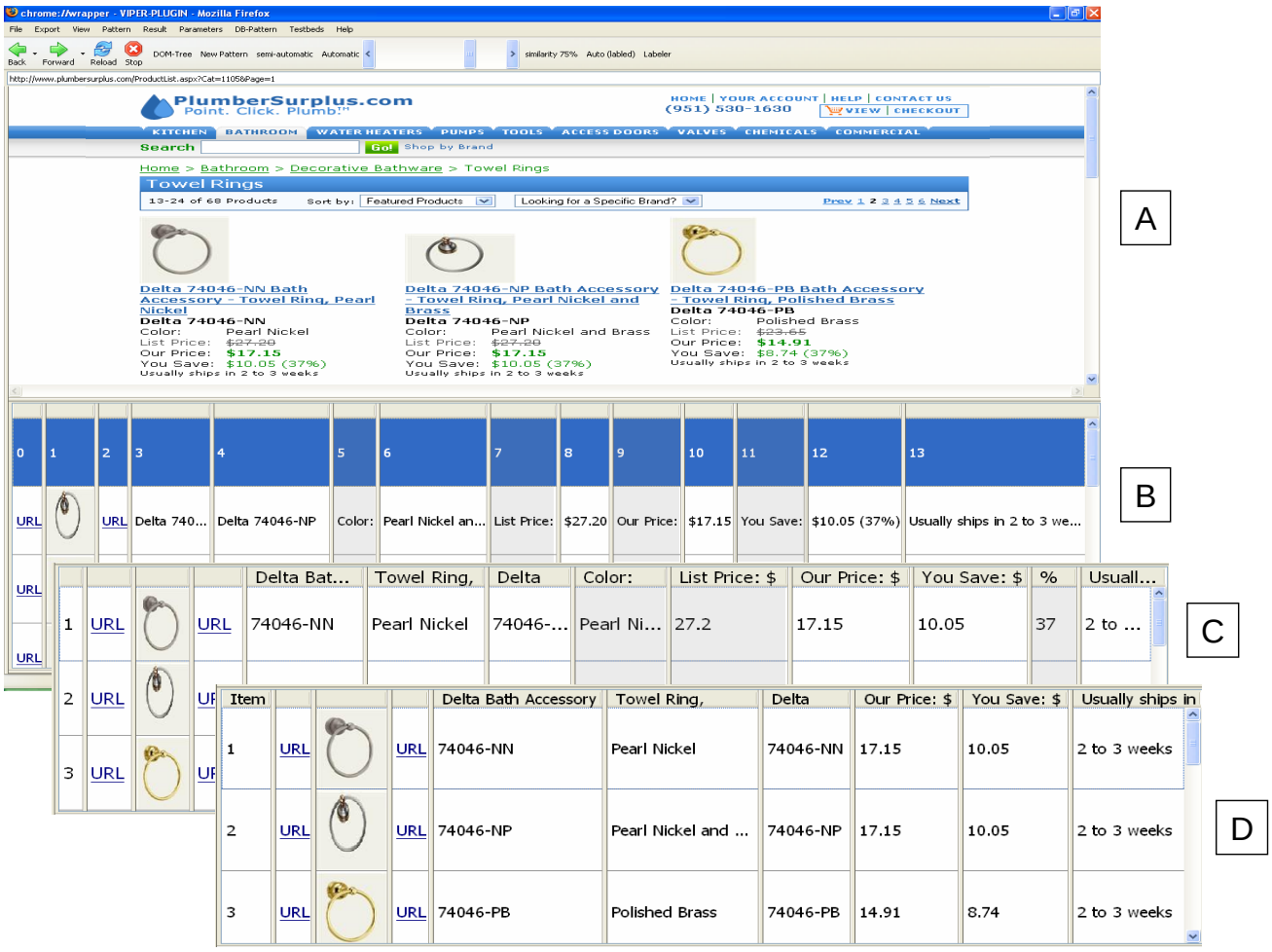

Figure 1: Example Web page with horizontally and vertically arranged data records and their corresponding tabular representation after the extraction and alignment process. Figure C) illustrates a compact table view of the data records with partially labeled columns. The final figure D) represents a personalized user view of the Web content where functionally dependent and redundant information has been removed.

with property $\sum_{k=1}^{D_j} p_{k,j} = 1$. Additionally we rank the values according to their frequency $n_{k,j}$ and denote by $v_{k,j}$ the k-th frequent value in column j.

3.2 Heuristic Mining

Applying heuristics is a straight forward strategy to mine simple properties in a table. Therefore in a first step we introduce some simple heuristics which identifies trivial properties. Based on these initial properties we are able to create more complex heuristics, subsequently. We made the following observations which have been directly hard coded into the table mining mechanism:

1. If the number of distinct values D_j in a column C_j is close to the column dimension, i.e. $D_j = N - \epsilon$ with some small $\epsilon \geq 0$, then we denote the column a key column. If $\epsilon > 0$ then the column represents "almost" a key and we call such a column a *soft key* column. It can be easily observed that the data items in the column determine the row with a high probability and hence the values in any other column. Therefore a (soft) key column is trivially statistically correlated with every other column in t and consequently we will not use these columns when searching for functional dependencies.

2. On the other hand if a column C_j contains only one type of data item, $D_j = 1$, then any other column functionally determines C_j in a trivial manner. Thus, we also omit this column when searching for functional dependencies.

Web data records often contain unique links resulting in soft/hard keys after the alignment process. We like to mention that sometimes more than only one column could become a key, for instance, the same unique link might be referenced multiple times inside a data record. Titles, pictures, etc. enclosed by anchor tags and pointing to the same location are common examples for such a situation. Deriving key columns from small sample sets is not statistically meaningful, therefore we always have to verify the key constraints afresh.

The second case often occurs if the data records contain fix data items appearing "next to", with respect to the HTML

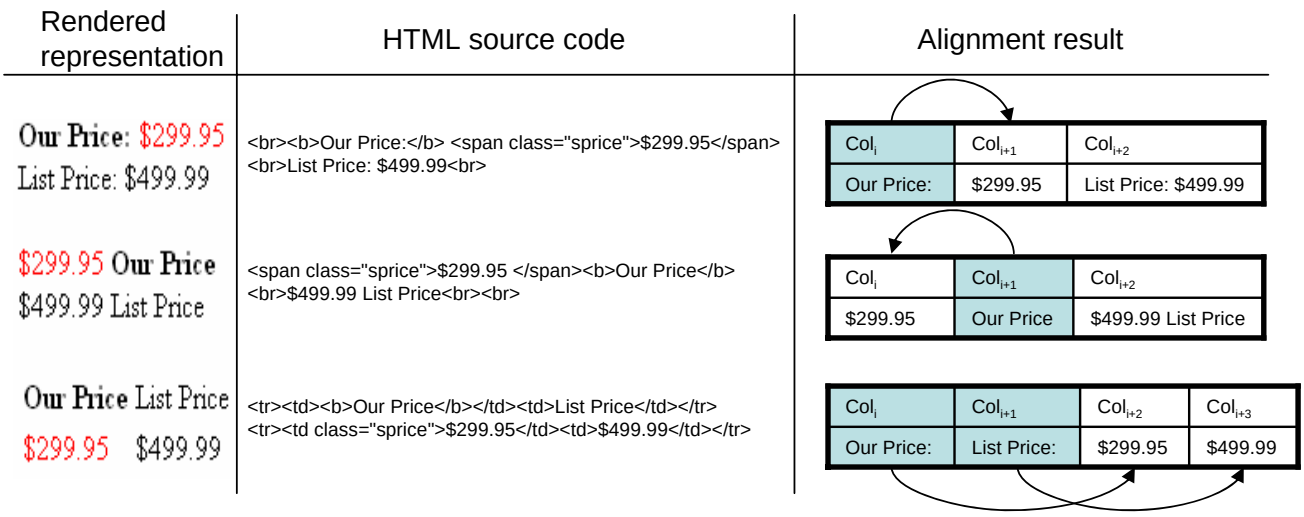

Figure 2: Heuristics to label data items have to rely on render information. We capture three different kinds of data item arrangements. The above figure illustrates these by showing the rendered browser representation (left) of their corresponding HTML tag sequence (middle) and aligned table representation (right) for each of the three cases. A column with fix content (gray), is a potential label and becomes assigned (arrows) with respect to the render information to the spatially closest column with variable content.

representation, variable data items. The interpretation of the resulting fix column depends on the layout structure. Figure 2 illustrates an example of three typical types of fix columns and their interpretation handled by our heuristics. Scanning top-down through the examples we distinguish the following label assignment strategies: left-to-right, right-to-left and in case of vertical orientations up-to-down assignment. We will not regard the fourth orientation down-to-up because of its rareness. The first orientation especially holds for documents encoded in (western) languages on account of the reading direction. In this case the aligned fix content (gray) can be easily assigned to the next column with presumable variable content (arrow). Otherwise, if the label is positioned on the opposite side of the variable content then we also have to assign the aligned fix content (gray) to its predecessor column. These label notations are often used for units. An indication which of these two assignment directions should be applied is realized by a decision tree. First we test whether the surrounding data items are in the same line, with respect to the render information. Provided that they are in the same horizontal line we next test if there exists a special separation character at the end of one of the strings, e.g., the colon in "Our Price:" from our first example. Of course we cannot apply this rule, if both of the surrounding data items are images. A final rule considers the complete row. Thereby the assignment direction which have been applied most times inside the row takes place. If non of these rules decide the assignment problem we leave the column untouched. A typical characteristic of the up-to-down assignment is that inside the aligned table a number of fix columns appear next to each other. To map the labels to the corresponding data items we search for the spatially closest located variable column in the render representation and thus assign the consecutive fix columns from left to right.

In our running example figure 1B) the gray marked columns at position 5,7,9 and 11 in the table have fix content ending with a colon. With the additional render information the heuristic assigns the content of the fix columns to their adequate right neighbor column as column label. Next each column becomes scanned for static text objects appearing in each row inside that column. For instance in figure 1B) each data item of column 8 and 10 contains the currency character "$" respectively in column 13 the coherent words "Usually ships in" are fix. Some objects on the other hand could split a column into two or more new columns. For instance in column 12 in figure 1B) each data item contains the fix characters "$,(,%,)". In our heuristic mining system we have specified some characters as column separator. In this case the pair of opening and closing parenthesis match these separators and therefore divide the column into two new columns. The resulting table generated by the heuristic mining algorithm which has become augmented by a leading enumeration column is depicted in 1C).

3.3 Dependency Mining

Automated database analysis and discovery of correlations between columns and rows from relations has received considerable interest [16, 8, 5, 14]. The most important ones in our context are functional dependencies (FDs) and arithmetic dependencies (ADs) between columns.

3.3.1 Functional Dependency Mining

With respect to relational data model we denote our table t *relation* and define:

The relation t *satisfies* the *functional dependency* (FD) $X \rightarrow Y$, if for every X-value x, $\pi_Y(\sigma_{X=x}(t))$ has at most one tuple. Here, X and Y denote subsets of attributes and π and σ denote the projection and selection operators, respectively.

For instance, in the data records in figure 1, the *You Save* price is determined by the *List Price* and *Our Price*. If there exists only some tuples violating the FD we speak of a *soft*

functional dependency (soft FD for short). We define a soft FD (denoted by $C_x \rightarrow C_y$), the finite set of attribute values of C_x which determines the values of C_y not with certainty, but merely with high probability. The discovery of unexpected but meaningful soft FDs seems to be an interesting and attractable goal because these kind of FDs provide valuable hints to many different information. The uncertainty of such a soft FD is based on the number of tuples that need to be removed from the relation t to become a hard FD: $\text{error}(X \rightarrow Y) := \min\{|r| \, | r \subseteq s \text{ and } X \rightarrow Y \text{ holds in } s \setminus r\} / |s|$. We next describe our set of statistical measurements to derive soft FDs.

When searching for dependencies in the table we have to check all possible column pairs against each other. After the previously mentioned heuristics have been executed we can use pruning rules to reduce the search space.

- *Text Constraints*: Prune columns which contain data items of mixed string type with more than 200 characters. This is typically the case for short text snippets describing a data record in more detail.
- *Pairing Constraints*: Prune column pairs where one of the participants fulfills a criteria mentioned previously in the heuristic part. [soft key or fix columns]

For the final set of column candidate pairs (C_l, C_k) and triples (C_l, C_k, C_m) we explore some classical statistical measures of association.

Our statistical dependency mining approach is divided into two stages:

In the first stage we use hash tables to mine functional dependencies. Hereby we generate a key value for the hash table function by combining at most two different columns, respectively their data items and insert it into the hash table. If we have a collision when inserting another row key from the combined columns into the hash table then we search for columns having equal data at the corresponding colliding rows. This technique directly checks the FD properties as defined above and enables us to discover FDs between pairs and triples of columns with arbitrary content. We have to mention that some of these FDs don't make sense. Therefore the user has to be consulted interactively which FD finally should be used.

In the second stage we inspect only those remaining column candidates which either consist of numeric data or categorical data like month names of a calender. Categorical data could be normalized by allocating a unique number to each possible text value. Finally, the candidates become discretized and normalized with the LUCS-KDD DN Software [15] leading to a greater data mining accuracy. Next start computing classical statistical measurements of association. One is the *Shannon entropy* which is a typical concept in information theory and expresses the randomness of data:

$$\mathrm{H}(C_j) = -\kappa \sum_{i=1}^{N} p_{i,j} \log p_{i,j},$$

where κ denotes a constant value which is just a choice of measurement units, we set $\kappa := 1.443$ and $p_{i,j}$ denotes the relative frequency. The entropy is maximal if the data items appear with the same frequency. In this case we have a uniform distribution of the data items contained in the column. We normalize the entropy to the interval $0 \leq \widehat{\mathrm{H}} \leq 1$ by multiplying $\log_2(N)$ to $\widehat{\mathrm{H}} := \mathrm{H} * \log_2(N)$. Next we search for an association between for instance a pair of columns by computing the the *chi-square distribution* (χ^2) and test if the null hypothesis is true.

$$\chi^2_{C_x,C_y} = \sum_{i=1}^{K} \sum_{j=1}^{N} \frac{(n_{i,j} - n_i n_j)}{n_i n_j}$$

Another prominent test which measure the strength of association of a cross tabulation is the *Cramer's V*-test. We define it by using the *mean-square contingency* (ϕ^2)

$$\Phi^2_{C_x,C_y} = \frac{\chi^2}{N}$$

resulting in:

$$\mathrm{V}_{C_x,C_y} = \sqrt{\frac{\Phi^2_{C_x,C_y}}{\min\{K-1, N-1\}}} \text{ with } 0 \leq V_{C_x,C_y} \leq 1$$

The values of V_{C_x,C_y} ranges from 0 (no association) to 1 (the theoretical maximum possible association). Due to the fact that in real-world datasets most column values are biased and therefore V_{C_x,C_y} never exactly equals zero, we consider the attributes of the pair (C_x, C_y) independent if $\mathrm{V}_{C_x,C_y} \leq \epsilon$ for some small $\epsilon > 0$. If this condition holds then we have sampled a column-value pair (C_x, C_y) from our table which is totally independent to each other.

Returning to our example in figure 1 and the resulting table after the heuristic mining 1C) we are now able to illustrate the dependency mining technique. First, the system discovers that the content of the column labeled "Color:" is redundant since it is totally determined by the column with label "Towel Ring" (totally equal data items). Additionally the system detects with statistical mining techniques that column C_u with labeled "List Price: $" and C_v labeled with "Our Price: $" determines column C_w labeled "You Save: $" or the column C_x labeled with "%". The user now can decide in a post-processing step which of these functional dependant columns are of interest. In the resulting table 1D) we can see that first the redundant "Color:" column disappeared and only column C_v and C_w are further displayed because they contain all the information needed to reconstruct the vanished columns C_u and C_x.

3.3.2 Arithmetic Dependency Mining

In the last dependency mining step we try to discover arithmetic dependencies between columns containing numeric data items. With respect to the computational complexity we limit the mining process to equations of the type

$$C_z = \lambda_1 C_x + \lambda_2 C_y \qquad\qquad C_z = \lambda_1 C_x$$

where $\lambda_i \in \mathbb{R}$ with $i \in 1, 2$. To get the relations between numeric columns we check the homogeneous system of linear equations for non trivial solutions. In case we have a solution and thus have linear dependent columns we try to describe each of these columns by a linear combination of at most two other columns as describes by the upper equations.

4. RULE GENERATION

Two language families that predated the (Semantic) Web, yet have been very useful for it, are positional languages based on Horn logic such as Prolog, and slotted languages with object-centered instance and class descriptions plus

$$\begin{array}{l}\texttt{tab}(?\hat{}s_1 \rightarrow f_1; \ldots; s_{i-1} \rightarrow f_{i-1}; s_i \rightarrow \mathbf{v_1}; s_{i+1} \rightarrow f_{i+1}; \ldots; s_N \rightarrow f_N) \\ \qquad\qquad \vdots \\ \texttt{tab}(?\hat{}s_1 \rightarrow f_1; \ldots; s_{i-1} \rightarrow f_{i-1}; s_i \rightarrow \mathbf{v_K}; s_{i+1} \rightarrow f_{i+1}; \ldots; s_N \rightarrow f_N)\end{array} \quad (1)$$

$$\texttt{tab}(?\hat{}s_1 \rightarrow f_1; \ldots; s_{i-1} \rightarrow f_{i-1}; s_i \rightarrow ?\mathbf{x}; s_{i+1} \rightarrow f_{i+1}; \ldots; s_N \rightarrow f_N) \texttt{ :- member}(?x, [v_1, \ldots, v_K]) \quad (2)$$

$$\begin{array}{l}\texttt{tab}(?\hat{}s_1 \rightarrow f_1; \ldots; s_{i-1} \rightarrow f_{i-1}; s_i \rightarrow ?x1; \ldots; s_{i+D-1} \rightarrow ?xD; s_{i+D} \rightarrow f_{i+D}; \ldots; s_N \rightarrow f_N) \\ \qquad\qquad \texttt{:- member}([?x1, \ldots, ?xD], [[v_{1,1} \ \ldots \ v_{1,D}], \ldots, [v_{K,1} \ \ldots \ v_{K,D}]])\end{array} \quad (3)$$

$$\begin{array}{l}\texttt{tab}(r_1\hat{}s_1 \rightarrow f_1; \ldots; s_{x-1} \rightarrow f_{x-1}; s_x \rightarrow v_1; s_{x+1} \rightarrow f_{x+1}; \ldots; s_{y-1} \rightarrow f_{y-1}; s_y \rightarrow w_1; s_{y+1} \rightarrow f_{y+1}; \ldots; s_N \rightarrow f_N) \\ \qquad\qquad \vdots \\ \texttt{tab}(r_K\hat{}s_1 \rightarrow f_1; \ldots; s_{x-1} \rightarrow f_{x-1}; s_x \rightarrow v_K; s_{x+1} \rightarrow f_{x+1}; \ldots; s_{y-1} \rightarrow f_{y-1}; s_y \rightarrow w_K; s_{y+1} \rightarrow f_{y+1}; \ldots; s_N \rightarrow f_N)\end{array} \quad (4)$$

$$\begin{array}{l}\texttt{tab}(r_1\hat{}s_1 \rightarrow f_1; \ldots; s_{x-1} \rightarrow f_{x-1}; s_x \rightarrow v_1; s_{x+1} \rightarrow f_{x+1}; \ldots; s_{y-1} \rightarrow f_{y-1}; s_{y+1} \rightarrow f_{y+1}; \ldots; s_N \rightarrow f_N) \\ \qquad\qquad \vdots \\ \texttt{tab}(r_K\hat{}s_1 \rightarrow f_1; \ldots; s_{x-1} \rightarrow f_{x-1}; s_x \rightarrow v_K; s_{x+1} \rightarrow f_{x+1}; \ldots; s_{y-1} \rightarrow f_{y-1}; s_{y+1} \rightarrow f_{y+1}; \ldots; s_N \rightarrow f_N)\end{array} \quad (5)$$

Table 1: Table compactification with POSL rules.

rules as in F-logic [9, 18]. Both have concise ASCII syntaxes, elegant semantics, and decent computational properties. Since these positional and slotted styles are often needed conjointly in the XML&RDF Web, they have been integrated in POSL[4]. In this paper, we use POSL as the 'human-oriented' syntax for tables (in the form of facts with the same relation name) as well as rules. For XML serialization, POSL is translated to RuleML [3] via the online POSL converter[5].

In its positional sublanguage, POSL uses a Prolog-like syntax except that variables are prefixed by a "`?`", with the anonymous variable written as a stand-alone "`?`". In its slotted sublanguage, POSL uses an F-logic-inspired syntax, where "*name*`->`*filler*" slots are separated, unordered, by a "`;`" infix, and slotted rests are made explicit with a "`!`" infix, usually followed by a variable. Facts and rule heads can be *anchored* by an OID (usually a URI) as a special 'zeroth' argument separated from further arguments by an up-arrow "`^`" infix.

Rules can be employed to *compactify* tables (without loss of information) or to *enrich* tables (with additional information). Compactification can affect table *rows* or *columns*. Enrichment is considered here for *columns* only.

Row compactification employs a rule to merge two or more rows that differ only in a few columns. This rule is constructed as follows. The rule head is formed from the merged rows by introducing fresh variables for data items that are different in the original rows. The rule body enumerates bindings of the tuple of these variables to tuples of data item values from the corresponding original rows. Enumeration can be done in several ways, one of which being the Prolog-like `member` predicate used here.

In particular, if there are K rows identical except for pairwise different data items v_j $(1 \leq j \leq K)$ in column s_i as formalised in table 1 with the POSL notation shown in (1) we are able to generate the rule (1). An example is guitar-describing rows that differ only in the color column.

[4][http://www.ruleml.org/submission/ruleml-shortation.html]
[5][http://www.ruleml.org/posl/converter.jnlp]

For $D > 1$ distinguishing columns s_i, ..., s_{i+D-1}, we can assume without loss of generality that these columns are adjacent in the POSL notation, obtaining the rule (3) notated in the table 1. An example with $D = 2$ might be amplifier-describing rows that differ only in the watt and price columns.

Column compactification employs rules to elide one or more columns that can be inferred from other columns. The head and body of these rules both refer to the `tab`le. The rules express specific column dependencies much like computable *functional dependencies* in relational databases. The computation is done by a function ϕ that can range from inserting a constant value, to copying a value, to using built-ins applied to a value. We consider here two important special cases.

Dependencies $C_x \rightarrow C_y$: The `tab`le show in (4) has the following form, where without loss of generality C_x is assumed to be on the left of C_y and $w_j = \phi(v_j)$ $(1 \leq j \leq K)$. This will be compactified to this `tab`le without column C_y as given in (5). The data items in column C_y will instead be computed when needed using a rule as follows. The body, for each row j $(1 \leq j \leq K)$ identified by an object variable `?r`, reads the slot filler v_j corresponding to column C_x and computes the functional value $w_j = \phi(v_j)$ from it. The head inserts, for those row objects `?r`, the results into the slot filler w_j corresponding to column C_y. In the POSL notation we assume without loss of generality that columns C_x and C_y are adjacent, obtaining the following rule, with the ϕ equation specialized to the appropriate function:

```
tab(?r^s_x->?v;s_y->?w!?z)
  :- tab(?r^s_x->?v!?z), ?w = φ(?v).
```

An example in figure 1C) is the column labeled "Our Price: $" which is multiplied with 0.63 the column labeled with "%". This could be expressed by $\phi = \lambda(a)a * 0.63$, where the resulting ϕ equation can also be written as a relation call `mult(?w,?v,0.63)`.

Dependencies $C_{x_1} \times C_{x_2} \rightarrow C_y$: The `tab`le has an obvious generalized form in which the two independent columns C_{x_1} and C_{x_2} together map to the dependent column C_y. With

the ϕ equation $w_j = \phi(v_{1j}, v_{2j})$, the rule becomes:

`tab(?r^`s_{x_1}`->?v1;`s_{x_2}`->?v2;`s_y`->?w!?z)`
`:- tab(?r^`s_{x_1}`->?v1;`s_{x_2}`->?v2!?z), ?w =` ϕ`(?v1,?v2).`

An example is the total column labeled "List Price: $" in table 1C) which is the sum of the column "Our Price: $" and "You Save: $" with $\phi = \lambda(a1, a2)a1 + a2$, where the resulting ϕ equation can also be written as a relation call `add(?w,?v1,?v2)`.

Column enrichment employs rules to add one or more columns that can be inferred from other columns. This is similar to column compactification except that inferred columns contain new information rather than data items elided from the original table. The information derived by rules constitutes extra virtual columns, i.e. the table itself is not changed.

Column enrichment can thus be regarded as starting with a table that results from column compactification and deriving the same rules. The two special cases discussed for column compactification above thus also apply here.

Without dependent columns available in the table for testing hypotheses, column enrichment requires other criteria for deciding whether new information may be of interest, ultimately via user interaction.

5. CONCLUSION

In this paper we presented a table mining extension of our fully-automatic information extraction system called ViPER. With the presented plugin realization of ViPER for Firefox a user is able to extract structured information from arbitrary Web pages while surfing the Web.

The system rearranges the information into a tabular representation and consolidates the resulting table by mining for structural and functional properties. Layered upon this, the Rex ViPER system compactifies and/or enriches the resulting tables by mining them for rules capturing structural and functional properties. At the current state of the system, some functional properties have to be interactively post-processed by the user. Finally, we serialize the compactification and enrichment rules in XML, specifically in RuleML.

By partially automating rule formation, Rex ViPER constitutes a major step towards Web Rules as envisaged by RuleML [4] and RIF[6]. This paper focused on the results of Rex ViPER, phase 1, where table and rule extraction perform visual structure recognition plus shallow string matching/parsing, and operate on a single document containing repetitive data records.

An obvious phase 2 enhancement will be the modular incorporation of a (controlled) English parser for those hypothetical table data items and rule arguments that are amenable to natural language processing as provided by projects such as Attempto Controlled English [7] and TRANSLATOR[7].

An independent phase 2 enhancement will be the extraction from, and integration of, tabular and rule information in multiple documents, as exemplified by our previous work on the New Brunswick Business Knowledge Base [12][8]. A further opportunity would be the mapping of imperfect correlations between table rows or columns to uncertain rules of the kind studied in Fuzzy RuleML[9].

Similarly, weights could be employed as in AgentMatcher [2] to sort columns according to their relative importance. Finally, background knowledge, e.g. in the form of ontologies about particular domains, could be utilized both for extraction and for rule generation, calling for rule-ontology integrations such as Datalog^{DL}[13].

[6][http://www.w3.org/2005/rules]

[7][http://www.ruleml.org/translator]

[8][http://www.ruleml.org/usecases/nbbizkb]

[9][http://image.ntua.gr/FuzzyRuleML]

6. REFERENCES

[1] M. Ball, H. Boley, D. Hirtle, J. Mei, and B. Spencer. The OO jDREW Reference Implementation of RuleML. In *Proc. Rules and Rule Markup Languages for the Semantic Web (RuleML-2005)*, pages 218–223. LNCS 3791, Springer-Verlag, November 2005.

[2] V. C. Bhavsar, H. Boley, and L. Yang. A Weighted-Tree Similarity Algorithm for Multi-Agent Systems in e-Business Environments. In *Proc. Business Agents and the Semantic Web (BASeWEB) Workshop*. To appear in: Computational Intelligence, Nov. 2004.

[3] H. Boley. Object-Oriented RuleML: User-Level Roles, URI-Grounded Clauses, and Order-Sorted Terms. In *Proc. Rules and Rule Markup Languages for the Semantic Web (RuleML-2003)*. LNCS 2876, Springer-Verlag, Oct. 2003.

[4] H. Boley, S. Tabet, and G. Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Proc. Semantic Web Working Symposium (SWWS'01)*, pages 381–401. Stanford University, July/August 2001.

[5] P. G. Brown and P. J. Haas. BHUNT: Automatic Discovery of Fuzzy Algebraic Constraints in Relational Data. In *Proceedings of the 29th VLDB Conference*, 2003.

[6] J. Frohn, R. Himmeröder, P. Kandzia, G. Lausen, and C. Schlepphorst. FLORID: A Prototype for F-Logic. In *International Conference on Data Engineering*, pages 583–583, 1997.

[7] N. E. Fuchs and U. Schwertel. Reasoning in Attempto Controlled English. In *PPSWR*, pages 174–188, 2003.

[8] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. In *SIGMOD*, Paris, France, June 2004.

[9] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM JACM*, 42(4):741–843, 1995.

[10] A. H. F. Laender, B. Ribeiro-Neto, A. S. D. Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *ACM SIGMOD Record*, 31(2):84–93, 2002.

[11] J. K. Lee and M. M. Sohn. The eXtensible Rule Markup Language. *Communication ACM*, 46(5):59–64, 2003.

[12] A. Maclachlan and H. Boley. Semantic Web Rules for Business Information. In *Proc. International Conference on Web Technologies, Applications, and Services (WTAS 2005), Calgary, Canada*. IASTED, July 2005.

[13] J. Mei, H. Boley, J. Li, V. C. Bhavsar, and Z. Lin. DatalogDL: Datalog Rules Parameterized by Description Logics, 2006. To appear in CSWWS2006.

[14] U. Nambiar and S. Kambhampati. Mining Approximate Functional Dependencies and Concept Similarities to Answer Imprecise Queries. In *Seventh InternationalWorkshop on theWeb and Databases (WebDB)*, 2004.

[15] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI Repository of machine learning databases, 1998.

[16] A. Pivk, Y. Sure, P. Cimiano, M. Gams, V. Rajkovi, and R. Studer. Transforming Arbitrary Tables into F-Logic Frames with TARTAR. In *Elsevier Science*, 2005.

[17] K. Simon and G. Lausen. Viper: Augmenting automatic information extraction with visual perceptions. In *Proceedings of the 2005 ACM CIKM Conference on Information and Knowledge Management*, pages 381–388, Bremen, GERMANY, November 2005. ACM Press.

[18] G. Yang and M. Kifer. Reasoning about Anonymous Resources and Meta Statements on the Semantic Web. In S. Spaccapietra, S. T. March, and K. Aberer, editors, *J. Data Semantics I*, volume 2800 of *Lecture Notes in Computer Science*, pages 69–97. Springer, 2003.