A Grammar Based Analysis of Column Header Categories for Web Tables

Mukkai Krishnamoorthy\(^1\)  Sharad Seth\(^2\)  Ramana Jandhyala\(^1\)  George Nagy\(^1\)

\(^1\)DocLab, RPI, Troy, NY, USA, \(^2\)CSE Department, University of Nebraska – Lincoln, USA
moorthy@cs.rpi.edu, seth@cse.unl.edu, ramanachakradhar@gmail.com, nagy@ecse.rpi.edu

Abstract

As part of a project to harvest semi-structured data from web tables, we describe an approach to extract an abstract representation of the column-header categories based on a context-free grammar for linear strings. The column-header structure is generally an XY-tessellation. The grammar provides a compact representation of infinitely many structural variations possible within column headers. Before parsing, the 2D column-header structure is converted to a linear string of its atomic cell labels and delimiters for the X and Y cuts. The acceptable strings represent a superset of admissible column-header structures from which the invalid ones are eliminated by performing geometric and lexical checks on the labels of the parse tree. Experimental results on web tables show that 80\% of the headers in the sample could be processed successfully using the grammatical approach.

1. Introduction

Our objective is to analyze the structure of row and column headers of a web table so as to obtain a layout-independent logical representation of the table. For example, given the header structure shown in Figure 1(a), the Wang notation tree (WNT) for the two categories, shown below in the indented form, captures the information essential to relate the entries in the table with the labels appearing in the table headers:

```
A
  B   A1   A2
  B1
  B2
```

Figure 1: Layout structures of table headers.

The WNT representation is useful in reformatting the table for different display or print devices. Moreover, it is potentially an important step in generation of table ontology \([10]\) and in providing metadata to improve the effectiveness of table-indexing search engines \([7]\).

Figure 2 shows a part of a web table from our sample. The row header \((\text{entity})\) appears in the first column of the headers. The entries \((\text{of other columns})\) in the top three rows define the column header with a hierarchical categorical structure. In this example there is only one category hierarchy with no explicit root category header.

Interested reader will find two extensive surveys of prior work on table analysis in references \([4]\) and \([12]\). Web tables must first be detected \([11]\) before the analysis proposed here can be
applied. Context free grammars have been used to characterize layout equivalent families of printed tables [5]. More recently, 2D graph grammars, with thousands of rules, have been devised to capture the geometric relationships between individual cells (boxes) of a table [2, 3]. In contrast, our context-free grammars require fewer than a dozen rules to analyze tabular headers. To the best of our knowledge there is no other work employing syntactic methods to analyze table headers.

2. Layout Structures of Table Headers

Figure 1(a) is a commonly used layout of headers for a table with the category A as the column header and category B as the row header. The labels of the root category in each case appear above their respective subcategories. Examples with missing or externally-found (e.g, within the caption) root-category labels are also common. Figure 1(b) shows an alternative layout in which the root-category labels appear to the left of the subcategory labels. We will call the two types of layout as Type 1 and Type 2 respectively. Hybrid layouts are also possible, where the two types are mixed within the same column or row header. Our focus in this paper is on column headers of Type 1 that satisfy the following constraints:

- They are laid out on a (possibly invisible) rectangular grid
- As a tiling of a rectangle by isothetic sub-rectangles, they define a subclass of XY-tessellations and, hence, they have a quasi-linear decomposition in the form of a XY-tree [8].
- In the case of two categories, the subcategory structure of the lower category is repeated for each subcategory of the higher category (see Fig 1(c))

We assume, for now, that any Type 2 or hybrid column headers have been converted to Type 1 in a preprocessing step. Further because of the X-Y symmetry that exists between column and row headers, it suffices to analyze only column headers.

3. Grammatical Approach and Results

The main contribution of this paper is a parsing approach to the analysis of the column header structure. To this end, the 2D column header is represented as a linear character string, which is parsed by a context-free grammar that accepts all valid structures. The string representation is possible because the column header is an XY-tessellation. This is illustrated in Fig. 3(a). The header shows the XY-decomposition both in the form of a tree and the corresponding character string in the figure. We use the term P-string to emphasize the use of parentheses as delimiters (curly for X-cuts and square for Y-cuts). This conversion, however, fails to preserve the category hierarchy of Type 1 column headers. For example, the subcategory headers A21 and A22 are separated in the string from their parent category header A2. An alternative cutting rule that overcomes this shortcoming can be stated as follows:

Make an X-cut only if it results in a leaf block; otherwise accumulate non-leaf block(s) until a leaf block (or the end of the parent block) is reached. Then Y-cut the accumulated block.

It is important to note that the new cutting rule already eliminates some XY-tessellations from consideration, e.g. the structure shown below:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: A web table from http://www.geohive.com/earth/ec_inet.aspx.
The performance is caused by the rigid assumptions about modifications to the structure of the table. This poor performance shows that it performs quite poorly. On 44 column headers sampled from a collection of 200 Excel and HTML tables from ten non-profit websites [9], G1 could accept only four without further modifications to the structure of the table. This poor performance is caused by the rigid assumptions about the column-header structure that often do not hold true. We summarize below several observed characteristics of real tables that are at odds with the model assumed in G1:

- Root category labels are commonly missing or span multiple rows in web tables while G1 assumes precisely one label that spans the width of the column header.
- It is also common to see one or more spanning rows at the bottom of the column-header: G1 does not allow for any such rows.

There are two ways to take care of this discrepancy between the real data and the model (a) Preprocess the data so that it fits the model, or (b) extend the model so as to be less rigid in its acceptance criteria. Following the first approach, we were eventually able to process all 44 column headers correctly. However, this solution requires manual intervention for each failing case. Instead, we present below an extended grammar for the column headers:

**G2 RULES**

1. S := A
2. S := {B}
3. B := c[X]B
4. B := c [X]B
5. X := cX
6. X := AX
7. X := A
8. X := c
9. X := cX
10. X := A
11. X := c

A comparison shows that both grammars are quite similar. Specifically, G2 retains six of the eight rules (Rules 1, 2, 5, 6, 7, and 8) of G1, modifies Rules 3 and 4, and adds three new rules. Two new rules (Rules 6 and 7) generate zero or more labels (symbol c) via the non-terminal C, while the new Rule 2 allows spanning rows at the bottom of the column header. The change in Rule 3 of G1 (from B := c [X]B to B := BB) does not affect the generative power of the grammar but was added to allow the automated parser generator to avoid conflicts. The change in Rule 4 of G1 (from B := c[X] to B := C[X]) allows category labels to be missing or span multiple rows.

When tested on the same 44 column headers, the results show a dramatic improvement in the performance over G1. Now, 35 (or 80%) of the P-strings were correctly parsed by the grammar by G2 compared to only 4 by G1.

**4. Additional Checks**

While G2 accepts most of the valid column structures found in our sample of real tables, it would also accept a large number of nonsensical structures that violate the Type 1 rules. This is because, as designed, it can independently check the validity of each category hierarchy but its context-free nature prevents it from validating geometric and lexical constraints that must exist between cells belonging to different category hierarchies. We illustrate how
additional checks can eliminate invalid structures through the example column header below:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>2000</th>
<th>2001</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEASON</td>
<td>Summer</td>
<td>Winter</td>
</tr>
</tbody>
</table>

The grammar $G_2$ will correctly identify the two column categories. However, it would give the same result even if the category cells were misaligned as follows:

<table>
<thead>
<tr>
<th>YEAR</th>
<th>2000</th>
<th>2001</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEASON</td>
<td>Summer</td>
<td>Winter</td>
</tr>
</tbody>
</table>

$G_2$ also has no way of recognizing the repeated labels in the bottom row, hence it would erroneously conclude that SEASON has four subcategories.

In summary, the layout rules of the column header impose additional geometric and lexical constraints that are not detected during parsing. For the above example these constraints are listed in Table 1. We use the function $E(B)$ to denote the linear extent of a leaf-level block B and the function $L(B)$ to denote its label. Further, we label the cells of the header top-to-bottom and left-to-right as C1 through C7 for specifying the constraints.

These constraints can be included in the parsing framework of parsing by attaching the geometric extent and label information to the parse tree as it is built. This enables a post-processor to verify the constraints and output the correct Wang notation tree for each category. We have

**Table 1: Additional geometric and lexical checks between category cells**

<table>
<thead>
<tr>
<th>Geometric Constraints</th>
<th>Lexical Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E(C5) \subseteq E(C2)$</td>
<td>$L(C5) = L(C7)$</td>
</tr>
<tr>
<td>$E(C6) \subseteq E(C2)$</td>
<td>$L(C6) = L(C8)$</td>
</tr>
</tbody>
</table>

built and verified such a post-processor for parse trees generated by $G_1$ and are in the process of doing the same for $G_2$.

5. Conclusion

As an extension of the presented approach, we envision a system that improves the range of acceptable tabular forms with use. Such a system would require closer integration of the parsing and WNT generation processes so that incremental changes to the rules of the grammar do not require writing a new WNT generator from scratch. This and other requirements of an adaptive system are the focus of our future research.

**References**


