Analysis and Taxonomy of Column Header Categories for Web Tables

Sharad Seth[2] seth@cse.unl.edu
Ramana Jandhyala[1] ramanachakradhar@gmail.com
Mukkai Krishnamoorthy[1] moorthy@cs.rpi.edu
George Nagy[1] nagy@ecse.rpi.edu

[1]DocLab, Rensselaer Polytechnic Institute, Troy, NY 12180, USA
[2]CSE Department, University of Nebraska – Lincoln, Lincoln, NE 68502, USA

ABSTRACT
We describe a component of a document analysis system for constructing ontologies for domain-specific web tables imported into Excel. This component automates extraction of the Wang Notation for the column header of a table. Using column-header specific rules for XY cutting we convert the geometric structure of the column header to a linear string denoting cell attributes and directions of cuts. The string representation is parsed by a context-free grammar and the parse tree is further processed to produce an abstract data-type representation (the Wang notation tree) of each column category. Experiments were carried out to evaluate this scheme on the original and edited column headers of Excel tables drawn from a collection of 200 used in our earlier work. The transformed headers were obtained by editing the original column headers to conform to the format targeted by our grammar. Forty-four original headers and their reformatted versions were submitted as input to our software system. Our grammar was able to parse and the extract Wang notation tree for all the edited headers, but for only four of the original headers. We suggest extensions to our table grammar that would enable processing a larger fraction of headers without manual editing.

Categories and Subject Descriptors
I.7.5 [**Document Capture**]: Document analysis.

General Terms
Algorithms

Keywords
Web tables, conversion, Wang notation, parsing, column-header grammar, table ontology

1. INTRODUCTION

The significance of developing capabilities for harvesting semi-structured data from web tables cannot be overestimated. Almost all nations post quantitative data such as the lengths of rivers or coast lines, heights of mountains, areas of lakes, population, age, ethnic origin, birth and death rates, immigration and emigration, education, employment, industrial production, commerce, and transportation. Canada Statistics (www.statcan.gc.ca), for example, has over 38 million series/vectors in over 2800 tables. Swiss Statistics currently has 50,033 tables, and India Statistics is even larger. In the US more specialized sites are maintained by various government departments: Agriculture, Energy, and Health. The CIA World Factbook and several international organizations like UNICEF and the World Bank offer tables of worldwide data. These sites are consulted frequently by the general public and by decision makers.

Although tables remain the accepted method for displaying data for human access, table layout and structure has been undergoing rapid change since our first studies twenty years ago [17]. Layout used to be governed primarily by human visual acuity and by page paper size (with rules promulgated by the US Government Printing Office and the University of Chicago manuals of style). However, advances in digital technology for page layout, typesetting, spread sheets, and browsers (e.g., scrolling, zooming, dynamic tables) have had significant effect on best practices of table construction. Here we focus on a large subset of web tables we call *canonical tables*. We postpone consideration of tables not laid out on a (perhaps invisible) rectangular grid, nested tables, concatenated tables, and tables containing graphics. Tables appear to be simple objects, but in fact the rules governing their layout and composition are recondite. It is now widely accepted that table understanding is a high-level cognitive skill that is not easily programmed [7]. Our focus here is the systematic analysis and formalization of geometric and topological table syntax.

Comprehensive reviews of two decades of research on table processing appear in [1, 28]. Researchers first developed algorithms for specifying cell location based on rulings [9, 18] or, in the case of un-ruled [6] and ASCII tables [12, 21], developed algorithms to determine typographic similarity of cell content and alignment [13, 14]. More recently researchers have addressed the information organizational aspects of tables, including associating content cells with header cells [8, 3, 2, 10]. They have devised methods to exploit the similarity of multiple tables from the same hidden-web source [26] and initiated analysis of augmentations such as table titles, captions, units, footnotes, and aggregates [23]. A proposal for an end-to-end system divides the table-understanding task into table detection, segmentation, function analysis, structural analysis, and interpretation, but was not

DAS '10, June 9-11, 2010, Boston, MA, USA

implemented and does not define which tables can and cannot be processed [25]. None of the methods that address web tables (e.g., [22, 4]) carry the analysis to the layout-independent multi-category level.

In the next section, we offer a taxonomy of tables. In Section 3 we revisit XY trees and show that a change in the order of decomposition from *breadth-first* to *depth-first* simplifies parsing a geometric representation of table headers using context-free grammars. Section 4 explains the geometric post-processing necessary to obtain the layout-free Wang category structure [27] from the parse trees. Section 5 describes an experimental evaluation of our approach on a sample of tables. Section 6 proposes extensions of our base grammar to accept a wider variety of tables. The Conclusions of Section 7 present, as is customary, all the work that still remains to be done to create a robust table interpretation system.

2. A (OUR) TAXONOMY OF TABLES

A canonical table is a rectangular tiling (*discrete isothetic rectilinear tessellation*) defined by the locations and types of all the *junction points* at which two orthogonal cell boundaries meet or cross. The number of tilings increases exponentially with the size of the grid. A 4×4 array has 70,878 different partitions. Some of these, called *XY tilings*, can be obtained by a divide-and-conquer method based on successive horizontal and vertical guillotine cuts. XY tessellations provide a structural representation of the rectangles obtained by horizontal and vertical cuts at alternating levels of a tree[1]. These tilings represent the miniscule but indispensable fraction of all tilings that are likely to be encountered as tables: The number of XY tilings relative to all possible tilings tends asymptotically to zero [15]. Figure 1 shows examples of rectilinear tessellations that are not acceptable as canonical tables. The conditions necessary for an XY table to be *canonical* are:

1. The tiling must be separable into four rectangular regions by a horizontal and a vertical ruling. Their point of intersection uniquely defines *stub-header*, *column-header*, *row-header*, and *data-cell regions* of a canonical table. The stub-header is empty.
2. Headers must span their immediate sub-headers, which must be (a) for column headers, in the row immediately below them and (b) for row headers, in the adjacent column to their right. This rule applies to every pair of header rows or columns that belong to the same category. There is no limit on the depth or width of the header hierarchies.
3. The header hierarchy for each category can be represented by a balanced rooted tree.
4. The number of lowest-level row and column sub-headers for the category trees nearest to the data cells must be equal to the number of rows and columns of data cells respectively.
5. Each data cell must be specified uniquely by a set of root-to-leaf category paths, one through each category tree.

[1] We originally proposed XY trees for page layout analysis [17, 15]. They have been periodically rediscovered and are also known by other names like *puzzle tree* or *tree-map* [22]. They transform a 2-D structure into two interlaced 1-D structures.

Some of these rules are violated in our sample. For example, a category root cell may be missing because it is obvious to the reader, or it may appear below or to the right of the leaves of the category tree. Headers may be separated from the data cells by spanning cells that. show units or are left blank. Sub-header cells may be split for no reason other than to balance a category tree. Since few tables found on the web meet all of the above conditions, we must manually or algorithmically convert non-conforming tables into canonical tables, or relax the requirement of a canonical table for the algorithm that extracts the Wang notation. In this paper, we present a solution following the first approach.

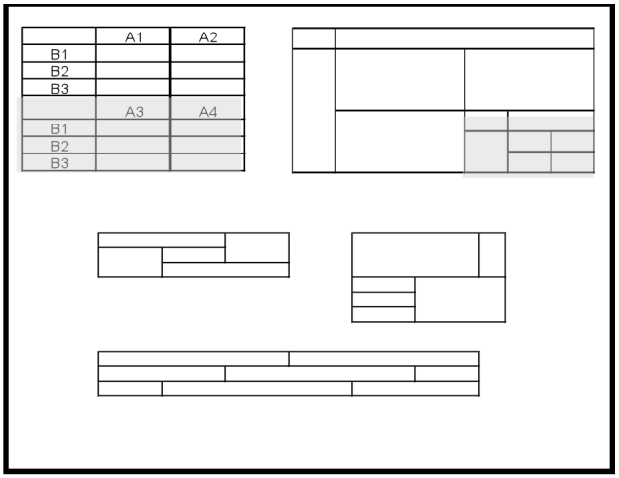


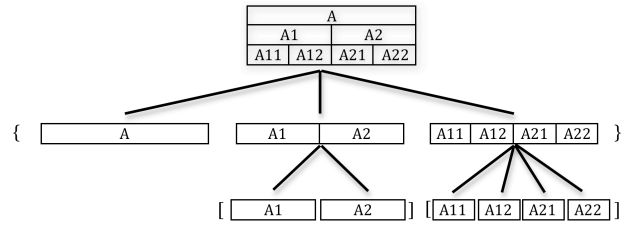
Figure 1. Rectilinear tessellations rejected as canonical tables: (a) a *concatenated table;* (b) a *nested table;* (c) a *non-XY tiling;* (d) and (e) two *non-tabular* XY tilings.

3. CUTTING ALGORITHM

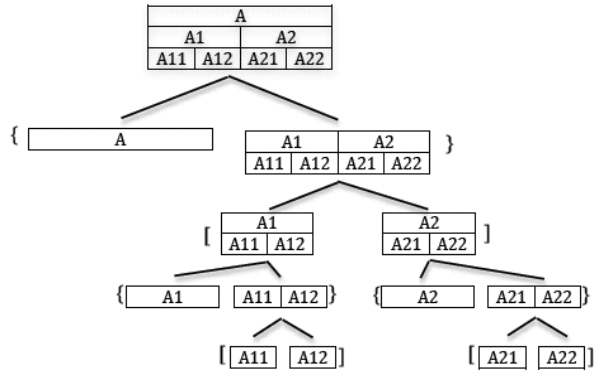
Decomposition of an XY-tessellation in terms of its component blocks can be carried out according to application-specific cutting rules [11]. The generic decomposition corresponding to the XY-tree has been found to be sufficient in many document segmentation algorithms [19, 20]. For analyzing the column (or row) header of a table using a context-free grammar, however, we have found the conventional XY-tree decomposition inconvenient. We illustrate this for a single-category column header in Figure 2 (a).

The column header block at the root of the tree is first cut into horizontal slices at every end-to-end (*guillotine*) cut in the horizontal direction. Each resulting block is then cut in the same way in the orthogonal (vertical) direction. This process of alternating the direction of cuts is iterated until only leaf blocks remain, and the process stops after the first vertical cutting. The parenthetical string (P-string) notation allows the decomposition tree to be represented in a linear form. We note that while the sub-categories A1 and A2 of A appear right after A within the vertical parentheses in the string, the sub-categories of A21 and A22 appear separated from them within the next group. This separation would complicate the parsing of the string to uncover the category hierarchy using a context-free grammar.

Alternate cutting rules, illustrated in Figure 2 (b), overcome this shortcoming. In the new rules, a horizontal cut is made if it results in a leaf block; otherwise, the non-leaf blocks are accumulated until a leaf block (or the end of the parent block) is reached. The accumulated block is then cut in the vertical direction using the standard XY-tree rules. The resulting tree and the P-string show that the sub-category labels are kept together with the label of their parent category by the new rules. Figure 3 shows the algorithm that cuts a column-header block according to the modified rules.



(a) P-String: { A [A1 A2] [A11 A12 A21 A22]}



(b) P-String: { A [{ A1 [A11 A12] } { A2 [A21 A22] }] }

Figure 2. Two decompositions of a table column-header and the resulting P-strings: (a) Breadth-first decomposition corresponding to the conventional XY-tree (b) Decomposition following column-header-specific rules for cutting blocks. For the P-string notation, the blocks obtained by horizontal cuts are surrounded by "{" and "}" and those obtained by vertical cuts are surrounded by "[" and "]". The decomposition is then uniquely represented by a left-to-right traversal of the leaf cells and these delimiters in the tree.

4. PARSING AND WNT GENERATION

The cutting algorithm described in the last section takes the place of lexical analysis step in compiling, during which the string to be parsed is scanned according to the rules defining the *tokens* of a programming language. The token stream produced by lexical analysis may or may not be a valid construct of the programming language. Therefore, the next step in compiling is parsing the token stream according to the rules of a context-free grammar. Earlier work has shown that context-free grammars can help characterize entire families of *layout-equivalent* printed tables [5].Unlike compiling, however, the acceptability of a P-string by the grammar is only a necessary and not a sufficient condition for the given structure to be a valid column-header. For sufficiency, we must perform additional geometric and lexical checks. In the next section, we show how these checks can be incorporated as a post-processing step to parsing of the P-string.

```
CutH(B) {
      print("{");
      repeat {
         while ( B is non-null and B1 = XY-cut(B, H) is a leaf) {
            print(Label(B1));
            B = B-B1}
         if (B is non-null) then {B2 = XY-cut(B,H); B = B-B2}
         while (B is non-null and Temp = XY-cut(B, H) is non-
         leaf) {
            B2 = B2 union Temp;
            B = B – B2}
         CutV(B2);
         } until B is null;
      print("}");
}

CutV(B) {
   print("[");
   repeat {
      B1 = XY-cut(B, V);
      if(B1 is leaf) then print(Label(B1)) else CutH(B1);
      B = B – B1;
   } until B is null;
   print("]");
}
```

Figure 3. The cutting algorithm.

4.1 Geometric and Lexical Checks

Consider Table 1 with the column header shaded gray. Note that the column header consists of two categories, "YEAR" and "SEASON". Each has two sub-categories but because "SEASON" appears below "YEAR", its two sub-categories are repeated for each sub-category of "YEAR", according to the commonly used rules of table layout. The column header is shown in Table 2, with its cells identified symbolically as c1 through c8 for ease of reference.

Using the cutting rules, we get the following P-string for the column header:

{ c1 [c2 c3] c4 [c5 c6 c7 c8] }

Table 1: Maximum Temperature

	YEAR			
	2000		2001	
	SEASON			
CITY	Summer	Winter	Summer	Winter
Montreal	35	11	36	2
Vancouver	28	18	29	19
James Bay	8	4	9	5

Table 2. Column header of Table 1

c1			
c2		c3	
c4			
c5	c6	c7	c8

The layout rules of the table impose geometric and lexical constraints on this two-category column header that are listed in Table 3. We use the function E(c) to denote the horizontal extent of a cell c and the function L(c) to denote its label. The lexical constraints L1 and L2 are easily checked by attaching the block label as an attribute of each token. Some of the geometric constraints can be built into the context-free grammar model. We illustrate this for the context free grammar for column headers, used in our earlier work [10]. The grammar is reproduced in Figure 4. The recursive Rule 3 of the grammar, along with the *base* condition defined by Rule 4, generate one or more column categories of the form "c[X]", where c is the category label and the non-terminal X generates its sub-category labels *below* it. Because the horizontal extent of the terminal c is the same as that of the non-terminal X, the grammar rules are sufficient to ensure the geometric constraints *within* each category. For example, the geometric constraints G1 and G2, of Table 3 are automatically satisfied by our grammar rules. However, rules like G3 and G4, involving blocks in different categories, cannot be checked because the two categories are generated independently by the grammar.

However, verifying that a given table belongs to an admissible class and then finding its Wang-notation tree, all within the framework of context-free grammars (CFGs), is attractive for several reasons:

1. CFGs provide a concise notation for specifying an infinite class of structures
2. A large body of theory and compiler tools have been built over decades for lexical analysis, parsing, and translation using CFGs
3. Only incremental modifications to an existing grammar are needed to substantially enlarge the class of acceptable structures, thus allowing a high degree of automation in translating web tables to the Wang notation.

Table 3. Layout rules

	Geometric Constraints		Lexical Constraints
G1.	$E(c2) \subset E(c1)$ $E(c3) \subset E(c1)$	**L1.**	$L(c5) = L(c7)$
G2.	$E(c5) \subset E(c4)$ $E(c6) \subset E(c4)$ $E(c7) \subset E(c4)$ $E(c8) \subset E(c4)$	**L2.**	$L(c6) = L(c8)$
G3.	$E(c5) \subset E(c2)$ $E(c6) \subset E(c2)$		
G4.	$E(c7) \subset E(c3)$ $E(c8) \subset E(c3)$		

Rules of Grammar G

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

Figure 4. A context-free grammar for column headers. The terminal symbols "{" and "}" indicate horizontal cuts while "[" and "]" denote vertical cuts. The terminal "c" represents a leaf-level cell that can be uniquely identified by its label and geometric attributes in the P-string as c1, c2, etc.

Therefore, we would like to integrate the geometric and lexical checks with the parsing of the column headers. We illustrate this for the example column-header block of Table 1 (reproduced in Figure 5), which can be parsed by grammar G.

Figure 6 shows the parse tree produced by a shift-reduce parser for grammar G working on the P-string of the column header. If the parsing fails for a given P-string, we can conclude that the input does not belong to the admissible class of column headers. However, if it succeeds, as in this example, we must still perform the geometric and lexical checks before generating the Wang notation from the parse tree. To facilitate the checks, we attach symbolic attributes to the nodes of the parse tree, as described in the figure caption. Now, the lexical and geometric checks are easily performed as a post process on the enhanced parse tree.

To perform the geometric checks, the number of categories in the input block must first be determined. As noted earlier, the non-terminal B is used in the grammar to generate one or more instances of categories using Rules 3 and 4. Each instance has the form "c[X]" where the terminal c corresponds to the root label of the category and the non-terminal X generates its subcategory labels. In the example parse, this happens in the steps corresponding to REDUCE_7 (where Rule 4 is used) and REDUCE_8 (where Rule 3 is used.) Hence, we can conclude that there are two categories. Further, because of how the geometric attributes are propagated, the horizontal extent of the node corresponding to REDUCE_8 includes the accumulated extents of both the category blocks, while the horizontal extent of the REDUCE_7 node corresponds to just the second category block. Therefore, by taking the difference E(REDUCE_8_Node) - E(REDUCE_7_Node), we can derive the extent of the first category block as (1:1,2:4). In general, by taking the difference in the extents of two successive REDUCE nodes nodes that generate categories in the parse tree, we can find the extent of each category block. Now, the geometric constraints between two adjacent categories C_i and C_{i+1}, where C_i is geometrically above C_{i+1} can be stated. These are specified in terms of the leaf nodes in the category tree of the two categories:

Row #				
1	YEAR			
2	2000		2001	
3	SEASON			
4	Summer	Winter	Summer	Winter
Column#:	1	2	3	4

Figure 5. The column header of Table 1 with rows and columns numbered to represent geometric extent of its blocks

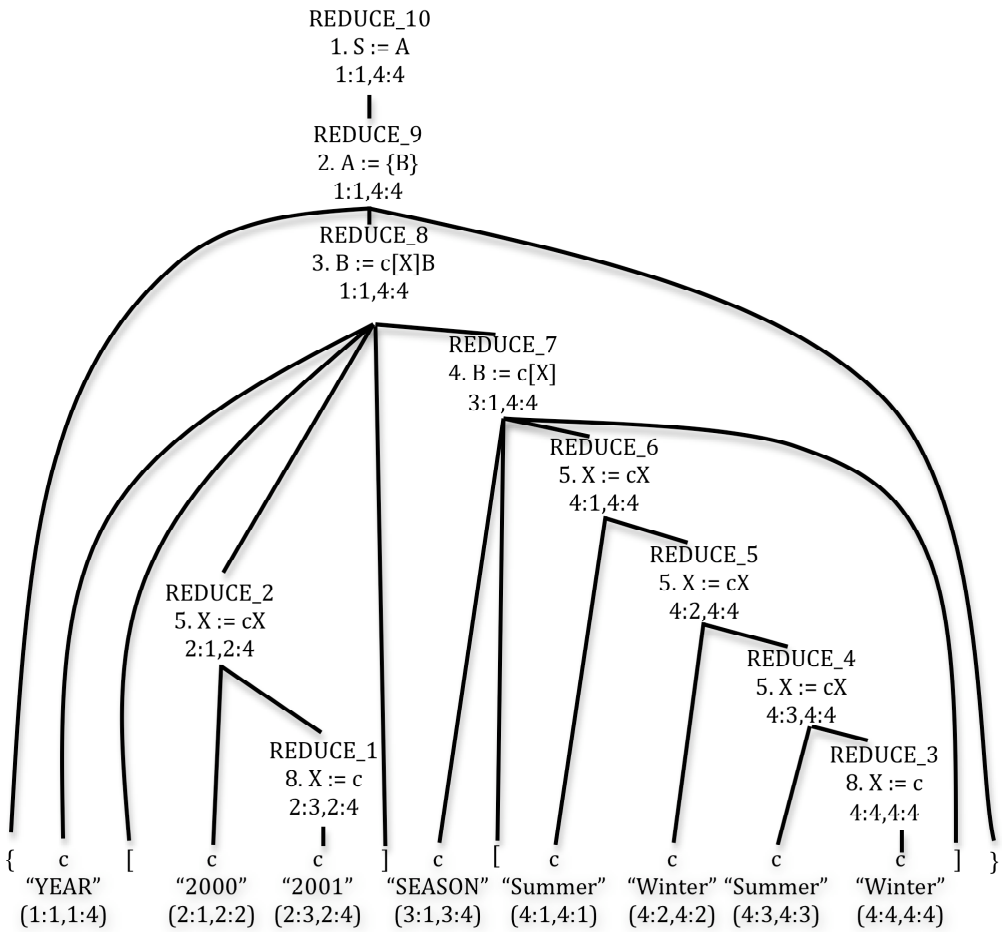


Figure 6: The parse tree for the column-header of Figure 2 according to the grammar G of Figure 4. The input P-string is shown at the bottom, along with the block labels and their extents associated with the terminal symbols c of the grammar, where the extent of a block is denoted by the *row*:*column* pair of its top-left and bottom-right cells. The shift-reduce parser scans the input string from left to right. For each input symbol it uses rules to determine whether the next symbol should be shifted on to a push-down stack or the top symbol(s) on the stack should be *reduced* according to a rule of the grammar. In this example, the first two reductions take place after the leftmost five symbols have been shifted on to the stack. In the first reduction (REDUCE_1) the terminal symbol c is replaced by the nonterminal symbol X using Rule 8 of the grammar. In REDUCE_2, this X along with the terminal c below it on the stack is replaced by X using Rule 5 of the grammar. During each reduction step, we also *propagate* the combined extent of the symbols being replaced to the node corresponding to the reduction. For example, for REDUCE_2, we combine the extents (2:1,2:2) and (2:3,2:4) and associate the extent (2:1,2:4) with this node. The acceptance of an input P-string is signaled by its reduction to the start symbol S in the final step of parsing.

1. (Geometric check) For every leaf cell c_j of category C_{i+1}, there must exist a leaf cell c_k of C_i, such that $E(c_j) \subset E(c_k)$. We denote this by saying that c_j is *covered by* c_k. For the example, if the extent of the cell "Summer" at (4:1,4:1), or the cell "Winter" at (4:2,4:2), in Figure 5, were to straddle the boundary between the cell "2000" at (2:1,2:2) and the cell "3000" at (2:3,2:4), this check would have failed.
2. (Geometric check) Let the number of leaf nodes of C_{i+1} covered by leaf node c_k of C_i be n. Then, the number of leaf nodes of C_{i+1} covered by every leaf node of C_i must be n. For the example, if the cell "2000" had covered two leaf cells of the category "SEASON" and the cell "2001" had covered three leaf cells of the same category, in Figure 5, this check would have failed.
3. (Lexical check) Not only must the number of leaf nodes specified in constrained 2 be the same but they must also be lexically identical. For the example, this check requires that both "2000" and "2001" cover the same labels ("Summer" and "Winter" in this case).

4.2 WNT Generation

If the geometric and lexical checks succeed, the program prints out a Wang notation tree [27] for each category in the indented form. The Wang notation highlights the layout-independence between the root header and their corresponding subcategories

for the column header region. Its representation in tree-like indented table-of-contents notation is visually intuitive. We illustrate the process of WNT generation for the running example.

After the geometric and lexical checks pass, the category hierarchy for the top category is derived by starting the parse tree traversal at REDUCE_8, where the "c[X]" part on the right-hand-side of Rule 3 generates the category tree reflected in the part of the substring "YEAR [2000, 2001]".

The WNT for the second category can be similarly found from traversal of the parse tree at REDUCE_7, however, because the second and lower categories contain repeated structures, we traverse only a non-repeated part of the structure, identifiable by the column span of a leaf sub-category of "YEAR". For example, because the leaf node "2000" of the top category extends from column number 1 to 2, we traverse only the "Winter" and "Summer" sub-category nodes of the "SEASON category. The final Wang notation tree in the indented form looks like:

YEAR
    2000
    2001
SEASON
    Summer
    Winter

5. EXPERIMENTS AND RESULTS

A sample of 51 tables, culled from the dataset described in [23], were imported into Microsoft Excel®. The table title, table caption, footnotes and notes were removed. The column headers of 44 of these tables were transformed into the format expected by our grammar. An example of such a transformation is shown in Fig. 7. One vertically concatenated table and six lists were excluded. Identical column headers from different tables were retained.

Using the algorithm described in Fig. 3, we constructed the P-strings for both the original headers and their transformed versions. P-strings are correctly parsed for all the transformed headers and their Wang notation tree is extracted. Without such manual preprocessing, P-strings of only four headers were accepted by the grammar. The results for the unedited case, shown in Table 4, indicate that a relatively high number of strings (33) were rejected. This validates our earlier conjecture (Section 2) that most real-world tables need some kind of editing to meet the conditions of the grammar.

An example of a column header in its native-state and after editing is shown, along with the intermediate P-string (with locations) and final output of the Wang category tree, in Fig. 8.

(Metropolitan Lima)	Rural Poverty (Sierra only)		
Total	Total	Relative	Indigence

(a)

VH			
(Metropolitan Lima) Total	Rural Poverty (Sierra only)		
	Total	Relative	Indigence

(b)

Figure 7. (a) Original column header, (b) Transformed header obtained by (i) adding virtual header VH, (ii) deleting the empty top row, (iii) merging headers defining the same column ("Metropolitan Lima" and "Total"), and (iv) merging isolated empty cells with the appropriate label "Rural Poverty (Sierra only)".

6. EXTENSIONS

The grammar G in Section 4 is intended to parse column headers of acceptable tables with rigid constraints. It assumes that there is no missing root-category label and further restricts the root category label to occur only in a single row. But missing root labels or repetitive labels are quite common in real tables as in Fig. 8, and are easily understood by humans from the contextual information. In this section we demonstrate how the grammar G can be extended to include a broader class of structures as follows:

1. The root header cell for any category can be missing (Fig. 9a).
2. The root label might span multiple cells (Fig. 9b).
3. There can be a "units" cell spanning the whole width of the column header at the bottom (Fig. 9b).

The grammar G1 below accommodates these extensions.

Rules of Grammar G1

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

S is the start symbol that generates all admissible strings in the grammar. The second rule is added to accommodate a single units cell at the bottom, represented by the terminal c. The rest of the grammar resembles G. Rules 4 and 5 of G1 are similar to Rules 3 and 4 with the non-terminal C substituting for the terminal c. The non-terminal C, which generates the root-category header, can produce any string of zero or more c's thus accommodating both the missing header and one spanning multiple rows. Rules 8 through 11 in G1 are identical to Rules 5 through 8. Thus, with only minor changes to the rules of the grammar, we are able to considerably extend the range of admissible structures.

Table 4. Results of attempting to extract Wang category tree before transforming the headers

Type of P-string	**# strings**
Accepted	4
Rejected	33
Ill-formed (improper nesting)	7
Total # of strings	**44**

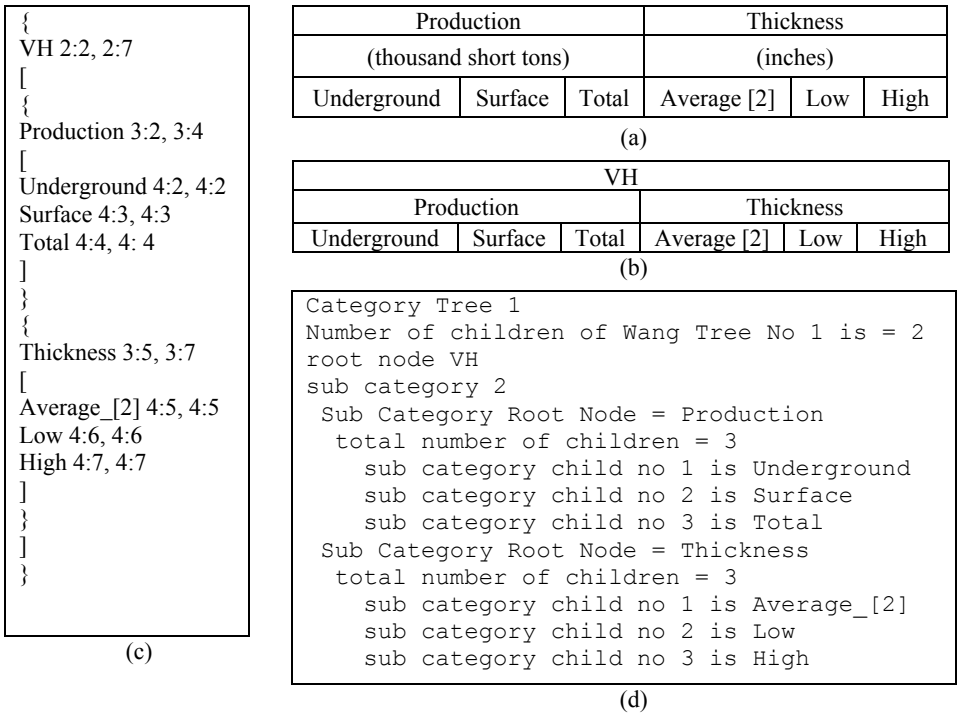


Figure 8: (a) Original header, (b) Transformed header, (c) P-string of the edited header, and (d) Indented Wang notation tree printed out by the program.

7. CONCLUSION

We presented a grammatical framework for parsing a linear-string representation of column headers of tables in a range of specified formats. Although the layout of the header represents a 2-D geometry, the layout constraints allow it to be decomposed as a form of XY tree with cutting rules adapted to the constraints. This simplified representation allows us to leverage compiler theory to build automatic parsers and attach actions to the parsing steps to write out the parse tree along with the geometric and lexical information contained in column header components.

In spite of the variety of column formats encountered in real web tables, we were able to edit into canonical form all the headers in our small sample that were not, and produce a correct Wang Notation tree automatically for every table. The approach presented here holds promises because many commonly found variations are handled easily by making only minor modifications to a base grammar.

We are currently exploring automation of the most time-consuming aspects of adapting an existing system, such as described here, incrementally to a larger class of structures, such as shown in Figure 9. Of particular interest are frequently occurring simple tables where the header is either just missing or left of the top subheadings. To study the learning curve involved in editing new table structures into an acceptable format, we also propose to run a larger experiment with "naïve" operators.

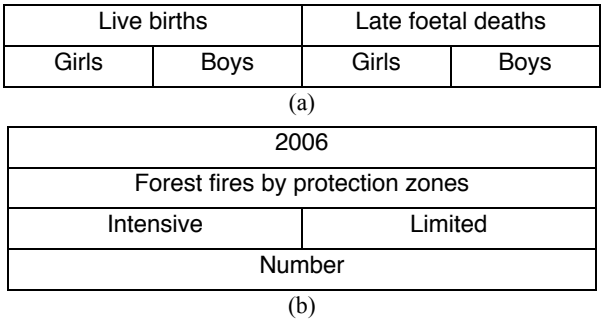


Figure 9. (a) A header without root header cell, (b) A header region with multi-cell root label at the top and a header-spanning label at the bottom.

ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under Grants # 044114854 and 0414644 and by the Rensselaer Center for Open Software. We are thankful to students Raghav Padmanabhan and William Silversmith of Rensselaer Polytechnic Institute for providing the data. We also gratefully acknowledge the influence of two decades of discussions about tables and ontologies with Prof. David Embley of Brigham Young University and Professor Dan Lopresti of Lehigh University.

REFERENCES

[1] Embley, D.W., Hurst, M., Lopresti, D., Nagy, G. 2006. Table Processing Paradigms: A Research Survey. Int. J. Doc. Anal. Recognit. 8 (2-3), Springer, Heidelberg, 66-86.

[2] Embley, D.W., Lopresti, D., Nagy, G. 2006. Notes on Contemporary Table Recognition. In: Procs. Document Analysis Systems VII, H. Bunke and A. L. Spitz, Eds., Nelson, New Zealand, LNCS # 3872, Springer, Heidelberg, 164-175.

[3] Embley, D., Tao, C., Liddle, S. 2005. Automating the extraction of data from HTML tables with unknown structure. Data Knowl. Eng., 54(1), July 2005, 3–28.

[4] Gatterbauer, W., Bohunsky, P., Herzog, Krupl, M., Pollak, B. 2007. Towards Domain-Independent Information Extraction from Web Tables. In Proceedings of the 16th International Conference on World Wide Web, Banff, Alberta, Canada, 71-80.

[5] Green, E. A., Krishnamoorthy, M., 1995. Model-based analysis of printed tables. In Procs. Third International Conference on Document Analysis and Recognition, (ICDAR), Montreal, Canada, pp. 214-217.

[6] Handley, J.C. 2001. Table analysis for multiline cell identification. In: Kantor, P.B., Lopresti, D.P., Zhou, J. (eds.) Proceedings of Document Recognition and Retrieval VIII (IS&T/SPIE Electronic Imaging), vol. 4307, San Jose, CA, 34-43.

[7] Hu J., Kashi R., Lopresti D., Nagy G., and Wilfong G. 2001. Why table ground-truthing is hard. In Proceedings of the Sixth International Conference on Document Analysis and Recognition, Seattle, WA, 129–133.

[8] Hurst, M. 2000. The Interpretation of Tables in Texts. Ph.D. thesis, University of Edinburgh.

[9] Itonori, K. 1993. A table structure recognition based on textblock arrangement and ruled line position. Proceedings of the Second International Conference on Document Analysis and Recognition (ICDAR'93), Tsukuba Science City, Japan, 765–768.

[10] Jandhyala, R. C., Nagy, G., Seth, S., Silversmith, W., Krishnamoorthy, M., Padmanabhan, R. 2009. From tessellations to table interpretation. In L. Dixon et al. (Eds.): Calculemus/MKM 2009, Springer-Verlag, Berlin, 2009, vol. 5625 of Lecture Notes in Artificial Intelligence, 422-437.

[11] Kanai, J., Krishnamoorthy, M. S., and Spencer, T., 1986. Algorithms for manipulating nested block represented images, SPSE's 26th Fall Symposium, Arlington VA, USA, pp.190-193.

[12] Kieninger, T., Dengel, A. 1998. A paper-to-HTML table converting system. In: Proceedings of Document Analysis Systems (DAS) 98, Nagano, Japan.

[13] Krüpl, B., Herzog, M., Gatterbauer, W. 2005. Using visual cues for extraction of tabular data from arbitrary HTML documents. Proceedings. of the 14th Int'l Conf. on World Wide Web, 1000-1001.

[14] Klink, S., Kieninger, T. 2001. Rule-based document structure understanding with a fuzzy combination of layout and textual features. International Journal of Document Analysis and Recognition, 4(1), 18-26.

[15] Klarner, D.A. Magliveras, S.S. 1988. Tilings of a Block with Blocks. Europ. J. Combinatorics, 9, 317-330.

[16] Krishnamoorthy, M., Nagy, G., Seth, S., and Viswanathan, M. 1993. Syntactic Segmentation and Labeling of Digitized Pages from Technical Journals. IEEE Transactions on Pattern Analysis and Machine Intelligence, 15(7), 737- 747.

[17] Kyriazis, G. 1990. Table Analysis. RPI DocLab Internal Report.

[18] Laurentini, A., Viada, P. 1992. Identifying and understanding tabular material in compound documents. Proceedings of the Eleventh International Conference on Pattern Recognition (ICPR'92), The Hague, 405–409.

[19] Nagy, G., Seth, S. 1984. Hierarchical Image Representation with Application to Optically Scanned Documents. In: Proceedings of the International Conference on Pattern Recognition VII, Montreal, 347-349.

[20] Nagy, G., Seth, S., and Viswanathan, M. 1992. A Prototype Document Image Analysis System for Technical Journals. IEEE Computer 25, July 1992, 10-22.

[21] Pyreddy, P., Croft, W.B. 1997. TINTIN: A System for Retrieval in Text Tables. In Proceedings of the Second ACM International Conference on Digital Libraries, New York, NY, 193--200.

[22] A. Pivk, P. Ciamiano, Y. Sure, M. Gams, V. Rahkovic, R. Studer. 2007. Transforming arbitrary tables into logical form with TARTAR. Data and Knowledge Engineering 60(3), 567-595.

[23] R. Padmanabhan, R. C. Jandhyala, M. Krishnamoorthy, G. Nagy, S. Seth, W. Silversmith. 2009. Interactive Conversion of Large Web Tables. Proceedings of Eighth International Workshop on Graphics Recognition, GREC 2009, Published by City University of La Rochelle, La Rochelle, France, July 22-23, 2009.

[24] Samet, H. 2006. Foundations of Multidimensional and Metric Data Structures. Morgan Kaufman.

[25] Silva, E.C., Jorge, A.M., Torgo, L. 2006. Design of an end-to-end method to extract information from tables. Int. J. Doc. Anal. Recognit. 8(2), Springer, 144–171.

[26] C. Tao and D.W. Embley. 2009. Automatic hidden-web table interpretation, conceptualization, and semantic annotation. Data & Knowledge Engineering, 68(7), July 2009, 683–703.

[27] X. Wang, "Tabular Abstraction, Editing, and Formatting," Ph.D Dissertation, University of Waterloo, Waterloo, ON, Canada, 1996.

[28] Zanibbi, R., Blostein, D., Cordy, J.R. 2004. A survey of table recognition: Models, observations, transformations, and inferences. International Journal of Document Analysis and Recognition, 7(1), 1–16.