From Tessellations to Table Interpretation

Ramana C. Jandhyala[1], Mukkai Krishnamoorthy[1], George Nagy[1],
Raghav Padmanabhan[1], Sharad Seth[2], and William Silversmith[1]

[1] DocLab, Rensselaer Polytechnic Institute, Troy, NY 12180, USA
[2] Computer Science and Engineering, University of Nebraska-Lincoln,
Lincoln, NE 68502, USA
`nagy@ecse.rpi.edu, seth@cse.unl.edu`

Abstract. The extraction of the relations of nested table headers to content cells is automated with a view to constructing narrow domain ontologies of semi-structured web data. A taxonomy of tessellations for displaying tabular data is developed. *X-Y tessellations* that can be obtained by a divide-and-conquer method are asymptotically only an infinitesimal fraction of all partitions of a rectangle into rectangles. *Admissible* tessellations are the even smaller subset of all partitions that correspond to the structures of published tables and that contain only rectangles produced by successive guillotine cuts. Many of these can be processed automatically. Their structures can be conveniently represented by X-Y trees, which facilitate relating hierarchical row and column headings to content cells. A formal grammar is proposed for characterizing the X-Y trees of layout-equivalent admissible tessellations. Algorithms are presented for transforming a tessellation into an X-Y tree and hence into multidimensional, layout-independent Category Trees (Wang abstract data types).

Keywords: document understanding, tables, rectangular tilings, X-Y trees, table grammars, Wang notation.

1 Introduction

Most quantitative data available in electronic form appears in the form of tables. We study formal aspects of web tables with a view to extracting their content. Various configurations of rectilinear tessellations defined on a grid can convey information in tabular form to human readers. In order to simplify the development of algorithms that recover the information from frequently occurring configurations automatically we construct a taxonomy of tabular layouts that may be considered equivalent from the perspective of table analysis.

Our work differs from earlier work w.r.t. (1) focusing on computer-constructed web tables rather than tables from scanned documents, (2) making use of commercial software to import web tables into a spreadsheet, (3) describing tables by X-Y trees and, most importantly, (4) facilitating content analysis by extracting the relationship of headers to content cells rather than only the geometric cell structure. This research is part of a larger project [1] to generate narrow-domain ontologies (e.g., for automobiles, obituaries, geopolitics) from semi-structured web data, which is itself a step towards realization of the Semantic Web [2,3]. Concentrating on tabular sources of quantitative information avoids some difficulties of natural language processing.

Comprehensive reviews of two decades of research on table processing appear in [4,5]. Algorithms were first developed for specifying cell location in terms of rulings or, in the case of unruled tables, according to the geometric alignment and typographic similarity of cell content. A recent proposal for an end-to-end system divides the task into table detection, segmentation, function analysis, structural analysis and interpretation, but was not implemented and does not define which tables can and cannot be processed [6]. None of the methods that address web tables (e.g. [7]), carries the analysis to the layout-independent multi-category level.

This paper formalizes the methods we used in an experiment on 200 tables randomly chosen from eight large web sites. The 200 tables were imported into Excel and edited into a form that could be processed algorithmically. The average size of the tables was 587 cells, and editing required on average 104 seconds [8]. *Augmentations* such as aggregates, annotations, footnotes and titles that are important componenents of most tables were also processed, but they are not included in the formalism presented here.

1.1 Rectangular Tessellations

A *discrete rectilinear tessellation,* or a *rectangular tiling,* is the partition of an isothetic rectangle into rectangles defined on an m x n lattice. The geometry of such a construct can be uniquely represented by the locations and types of all its junction points, i.e., points at which two non-collinear lines meet or cross. The number of tilings, $N_{all}(m) \equiv N_{all}(m,m)$, increases exponentially with the size of the grid. A quick count reveals that even a 4x4 grid has 70,878 different partitions. Some of these, called X-Y-tessellations, can be obtained by a divide-and-conquer method based on successive horizontal and vertical guillotine cuts. Klarner and Magliveras proved that the number $N_{xy}(m)$ of X-Y-tilings decreases quickly with the size of the grid [9]. Although $N_{xy}(4) = 68,480$, which does not differ in order of magnitude from 70,878,

$$\lim_{m \to \infty} N_{xy}(m) / N_{all}(m) = 0 .$$

Figure 1 shows a simple X-Y-tessellation, and Figure 2 shows tilings that are not X-Y-tessellations. In the VLSI literature these are known as *nonslicing structures* [10]. It is known that horizontal and vertical polar graphs (that are duals of each other) can be drawn for any rectangular tiling, and that for a *slicing structure* (X-Y-tessellation) the polar graphs are series parallel. The concept of polar graph goes back to a 1940 paper on the dissection of rectangles into squares [11].

Polar graphs abstract away the geometry of rectangular tilings but preserve the adjacency relationship between the tiles in the horizontal and vertical directions. X-Y

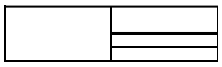


Fig. 1. A simple X-Y tessellation

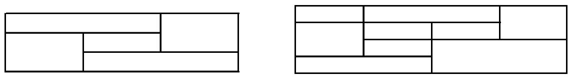


Fig. 2. Two non-X-Y tessellations

trees similarly abstract the geometry X-Y of tessellations by providing a *structural* representation of the rectangles obtained by horizontal and vertical cuts at alternating levels. Such partitions can be represented by X-Y trees that we originally proposed for page layout analysis [12, 13]. They have been periodically rediscovered and are also known by other names like *puzzle tree* or *treemap* [14]. They transform a 2-D structure into two interlaced 1-D structures, thereby facilitating analysis. Figure 3 shows two X-Y-tessellations defined on a 4 x 4 lattice that are geometrically different but are both represented by the X-Y tree shown on the right. We don't know the number of structurally different X-Y tessellations, $N_{S,xy}(m)$, but it clearly is much smaller than the number of (geometrically) different X-Y tessellations $N_{xy}(m)$. The transformation of an X-Y-tessellation to an X-Y tree is discussed in Section 2.

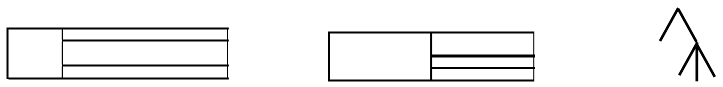


Fig. 3. Two geometrically different but structurally identical tessellations

1.2 Web Tables

The layout of tables for the presentation of information is dictated by convention. The Chicago Manual of Style [15] and the US Government Printing Office Style Manual [16] both have lengthy chapters describing these conventions. All tables have a *stub*, *column headings*, *row headings*, and *data cells*. Several common layouts are illustrated in Figure 4. Tessellations that correspond to such layouts are called *admissible tessellations* or table candidates because the location of each data cell is specified by a set of hierarchical row and column headings.

Many tables that appear in the literature do not strictly follow conventions yet are readily understandable by their intended readers. For example, a common occurrence is

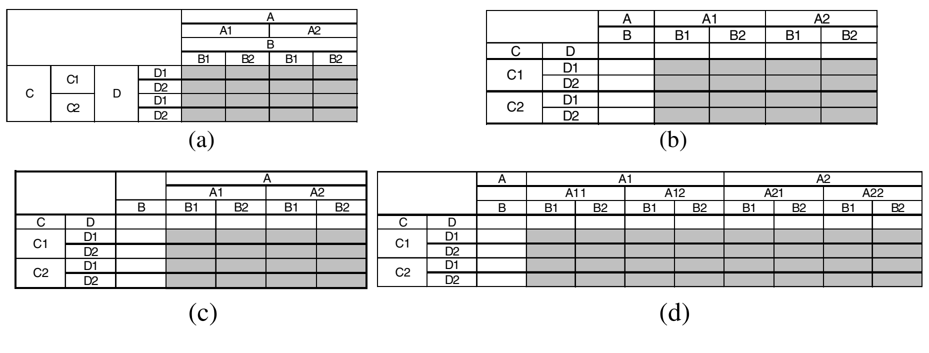
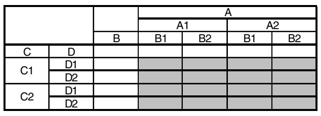


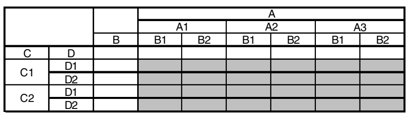
Fig. 4. Common table layouts. The blank top-left area is the *stub*. Only the *column* and *row headings* are labeled. The gray areas are content (*delta*) cells. Combinations of (a) for columns and (b) for rows are popular. (c) and (d) are more unusual hybrids.

the absence of a *root*, or spanning heading, for a category. Let us call the mathematically indefinable and unknown number of human-understandable tables $N_{T,S,xy}(m)$. We propose to process tables in this category by interactively transforming them into a smaller set of *admissible* tables that can be formally described and algorithmically analyzed. The number of admissible tables is $N_{A,S,xy}(m)$.

For the purpose of algorithmic analysis we need consider only *layout-equivalent* admissible table candidates that do not differ in the number of categories, but only with respect to the depth of their heading hierarchies, or the number of rows and columns, as do the examples in Figure 5.



			A			
			A1		A2	
		B	B1	B2	B1	B2
C	D					
C1	D1		▨	▨	▨	▨
	D2		▨	▨	▨	▨
C2	D1		▨	▨	▨	▨
	D2		▨	▨	▨	▨



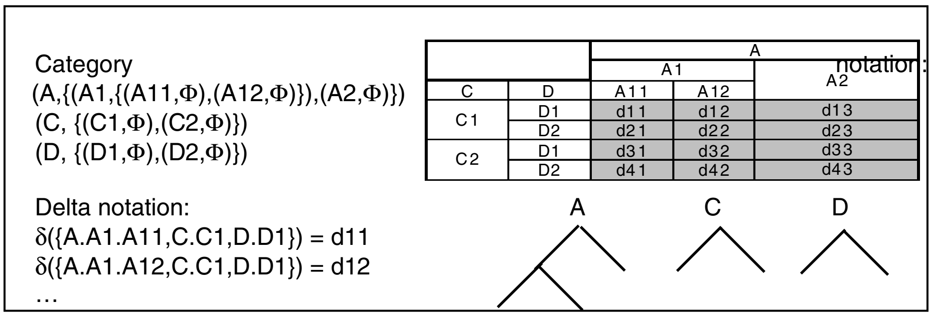
			A					
			A1		A2		A3	
		B	B1	B2	B1	B2	B1	B2
C	D							
C1	D1		▨	▨	▨	▨	▨	▨
	D2		▨	▨	▨	▨	▨	▨
C2	D1		▨	▨	▨	▨	▨	▨
	D2		▨	▨	▨	▨	▨	▨

Fig. 5. Layout equivalent tables. The blank areas must be empty. Gray areas contain data.

Context-free grammars can help to characterize entire families of layout-equivalent admissible tessellations, as first demonstrated in [17, 18, 19] and revived here in Section 3. A few such families account for the vast majority of tables encountered in books, journals, and the web. The number of different layout-equivalent admissible table candidates is $N_{L,S,xy}(m)$. We cannot yet process automatically all structurally equivalent admissible tables, therefore $N_{L,S,xy}(m) < N_{A,S,xy}(m)$.

X-Y trees represent only the physical layout of a table, which can be modified to suit page size or column width, or display characteristics. The first step in *understanding* a table is to analyze its logical structure, which is independent of the presentation aspects. Interpretation requires understanding the relationship between *headings* and *content cells*. An abstract data structure for this purpose was proposed by Wang in 1996 [20]. It represents headings in terms of category trees (*labeled domains*), whose Cartesian product provides the paths to every content cell (called *delta cells*). The number of categories in a table is called its *dimensionality*. Figure 6 displays the category trees for a simple table. The *size* of the table is the product of the number of rows and columns of delta cells, and it is also equal to the product of the number of leaf nodes in the category trees. An algorithm for extracting the Wang Notation from the X-Y trees is presented in Section 4.

Labeled table candidates for which Wang Notation exists are called Well Formed Tables (WFT). They are only a subclass of tables encountered in practice. However,



Category
(A,{(A1,{(A11,Φ),(A12,Φ)}),(A2,Φ)})
(C, {(C1,Φ),(C2,Φ)})
(D, {(D1,Φ),(D2,Φ)})

		A		
		A1		A2
C	D	A11	A12	
C1	D1	d11	d12	d13
	D2	d21	d22	d23
C2	D1	d31	d32	d33
	D2	d41	d42	d43

Delta notation:
δ({A.A1.A11,C.C1,D.D1}) = d11
δ({A.A1.A12,C.C1,D.D1}) = d12
...

Fig. 6. Wang notation for the categories and data cells of a simple 3-category table

most such tables can be transformed to WFT format with little effort. Figure 7 shows a table that is not well formed, and its WFT equivalent, obtained by the addition of *virtual headings*. The headings shown are sensible, but any arbitrary labels would do for the Wang notation.

Analyzing the logical structure of a table is necessary but by no means sufficient for understanding it. Understanding most tables requires considerable context and knowledge that extends far beyond the table under consideration. There is ample evidence that automating table understanding, or even merely verifying claims to this effect, is very difficult [21, 22, 23].

Table I Maximum temperature

	2000		2001		2002	
	Summer	Winter	Summer	Winter	Summer	Winter
Montreal	35	11	36	2	37	13
Vancouver	28	18	29	19	30	20
James Bay	8	4	9	5	10	6

Table I Maximum temperature

	YEAR					
	2000		2001		2002	
	SEASON					
CITY	Summer	Winter	Summer	Winter	Summer	Winter
Montreal	35	11	36	2	37	13
Vancouver	28	18	29	19	30	20
James Bay	8	4	9	5	10	6

Fig. 7. Top: Rootless categories: not an admissible table. Bottom: Virtual headings added to obtain an admissible configuration that is also a WFT.

As mentioned, our project is the front end of a larger undertaking that endeavors to create narrow-domain ontologies by combining information from web tables [1, 24, 25]. Suppose, for instance, that we process the left-hand table in Figure 8 and include it into the ontology. Then when we encounter the right-hand table we hope to be able to learn that the *hepth* of *goldam* is *320 gd* [26]. Our current plans to build interactive software for harvesting web tables based on the formalisms described above are outlined in Section 5.

Our approach to the gradual automation of table processing is based on the following inequalities, which show that useful tessellations are only a very small fraction of all possible tessellations. The various classes of tables are illustrated in Fig. 9.

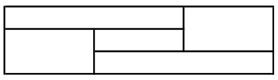
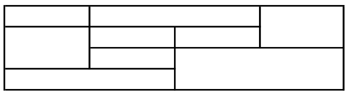
$$N_{L,S,xy}(m) < N_{A,S,xy}(m) < N_{T,S,xy}(m) << N_{S,xy}(m) << N_{xy}(m) << N_{all}(m).$$

fleck	**gonsity** (ld/gg)	**hepth** (gd)
burlam	1.2	120
falder	2.3	230
multon	2.5	350

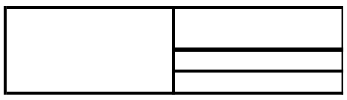
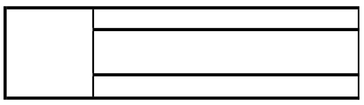
goldam	1.3 ld/gg	320 gd
falder	2.3 ld/gg	230 gd
elmer	2.9 ld/gg	350 gd

Fig. 8. Two tables with overlapping information

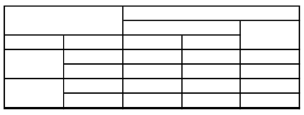
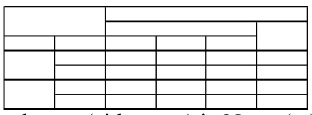
$N_{L,S,xy}(m) < N_{A,S,xy}(m) < N_{T,S,xy}(m) << N_{S,xy}(m) << N_{xy}(m) << N_{all}(m)$.

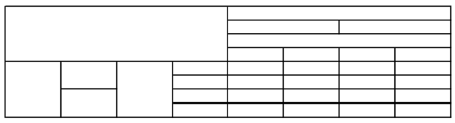
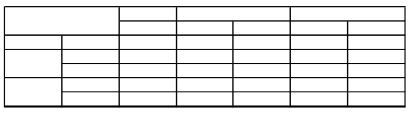
Tessellations in $N_{all}(m)$ but not in $N_{xy}(m)$ (no X-Y tree representation).

Both tessellations in $N_{xy}(m)$, but only one (either one) in $N_{S,xy}(m)$
These tessellations are *structurally equivalent*, but they are not *admissible* and therefore not table candidates.

Both tessellations in $N_{S,xy}(m)$, but only one (either one) in $N_{A,S,xy}(m)$.
These tables are *layout equivalent*, but we cannot yet parse them, and therefore they are not in $N_{L,S,xy}(m)$.

They are two different admissible tessellations in $N_{L,S,xy}(m)$. We can parse both.

				A			
				A1		A2	
				B			
				B1	B2	B1	B2
C	C1	D	D1				
			D2				
	C2		D1				
			D2				

				A			
				A1		A2	
				B			
				B1	B1	B2	B2
C	C1	D	D1				
			D2				
	C2		D1	X	X		
			D2				

The table on the left is a four-dimensional Well Formed Table. The table on the right is not a WFT, because the category paths {A,A1; B,B1; C,C2; D,D1} cannot distinguish between the two content cells marked X.

Fig. 9. Discrete rectangular isothetic tessellations. Our taxonomy does not include human-readable tables to which Wang Notation is inapplicable. The top table without a row header in Fig. 7 is certainly in $N_{T,S,xy}(m)$, but we cannot formally define *all* human-readable tables.

2 Tessellations to X-Y Trees

As discussed above, the X-Y tree is an economical representation of layouts that are of interest in table processing. Similar table layouts yield X-Y trees with similar structures. We can identify tables from which we can algorithmically extract Wang Notation. We shall also attempt to characterize families of inadmissible table structures that can be converted into admissible structures by a few editing steps. We expect to be able to automate such frequently used editing protocols.

The horizontally and vertically ordered lists of the *indices* of the junction points of a tessellation are not sufficient to derive the corresponding X-Y tree, although the combination of pre-order and post-order traversals uniquely characterizes general trees. The lists do not characterize the adjacency topology of the tessellations sufficiently for table analysis. Figure 10 illustrates tilings that are not differentiated by the structure of their X-Y trees, and different tilings with identical lists. For table analysis, the lists and trees must contain additional data, i.e., the vertical or horizontal location of the junction points and the type (e.g. NE-corner, T-connection, crossing) of each junction. This allows checking the alignment of cuts in separate subtrees.

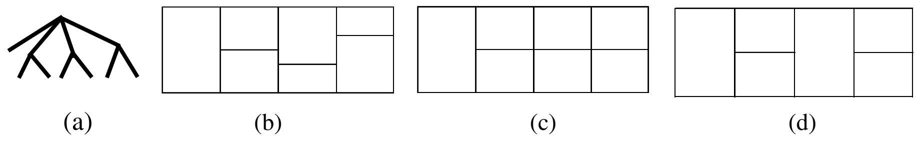


Fig. 10. The vertical cut X-Y tree (a) is the same for tilings (b) and (c), but not for (d). However, tilings (c) and (d) have the same lists of junction point coordinates.

The recursive algorithm EX2XY obtains the X-Y tree for any tessellation for which the tree exists. We use it to transform web tables imported into Excel. For portability, EX2XY produces an XML file. It takes the junction-points data for an X-Y tessellation and produces a fully-parenthesized representation for it, which can either be printed (saved) as a linear string of leaf-block labels and the two kinds of parentheses. It can also include geometric information attached to the labels and the left parentheses (of either type) in an internal data structure. The latter representation is useful for geometric and lexical checks.

The workhorses of the algorithm are two functions CutV(R) and CutH(R) which cut the given rectangle R, respectively, in vertical and horizontal directions. R may be specified as (x1,y1,x2,y2), where (x1,y1) are their top-left and (x2,y2) are the bottom-right junction points.

CutV and CutH return the first (leftmost or topmost) sub-rectangle of R, obtained by a guillotine cut. In the example rectangle of Fig. 11, CutV((1,1,4,4)) would return the sub-rectangle (1,1,3,4) and CutH((1,1,4,4)) would return the sub-rectangle (1,1,4,2). The cut may be *trivial* or *degenerate*, e.g for R = (2,1,3,2) in the example, CutV(R) = CutH(R) = R.

CutV and CutH are used in a pair of procedures, P1 and P2, which call each other recursively (Fig 12). P1 cuts a given rectangle vertically, submitting the leftmost sub-rectangle to P2 for horizontal cuts. Similarly, P2 cuts a given rectangle horizontally, submitting the topmost sub-rectangle to P1 for vertical cuts. The main procedure calls P1 with the outermost rectangle (1,1,4,4) for vertical-cut first, and P2 for horizontal-cut first.

Although most of our illustrations contain simple examples created directly in Excel, Figure 13 shows part of an actual web table, its Excel version created by the built-in IMPORT functionality, and its appearance after editing.

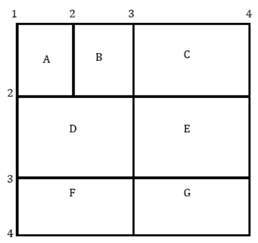
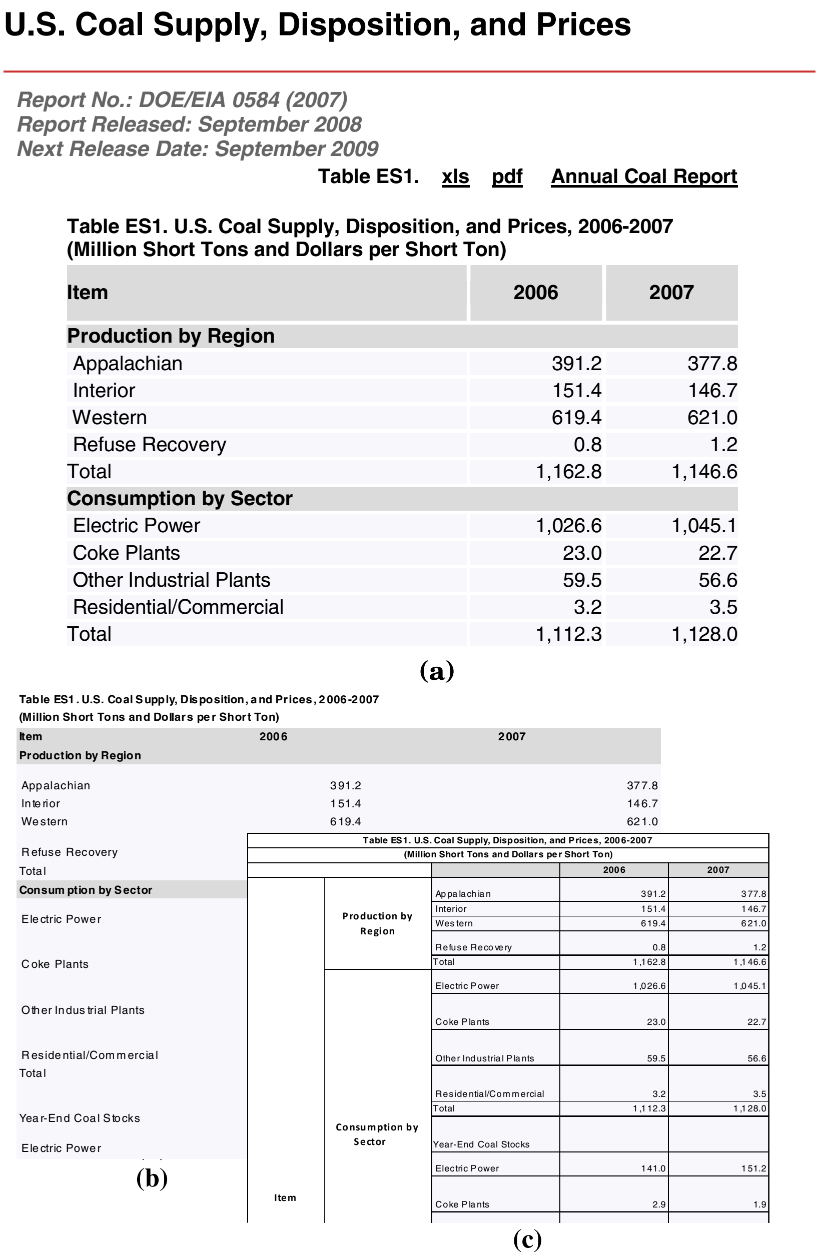


Fig. 11. A simple example to illustrate algorithm EX2XY

```
P1(R); {
    Declare S: rectangle

    S = CutV(R);
    if (S == R) then {if CutH(R) == R) then {print(label(R)); return} }
    /* Also, attach coordinates of R with the label of R */

    else /* have a non-trivial cut */
    {
        print("[");         /* Also, attach coordinates of  R with this "[" */

        Loop {
            P2(S);   /* H-Cut S */
             R = R-S;
             S = CutV(R);
         } until S==R;

         P2(S);             /* H-Cut the last rectangle */

        print("]")
    }
}

P2(R); {
    Declare S: rectangle

    S = CutH(R);
    if (S == R) then {if CutV(R) == R) then {print(label(R)); return} }
    /* Also, attach coordinates of R with the label of R */

    else /* have a non-trivial cut */
    {
        print("{");         /* Also, attach coordinates of  R with this "{" */

        Loop {
            P1(S);   /* V-Cut S */
             R = R-S;
             S = CutH(R);
         } until S==R;

         P1(S);             /* V-Cut the last rectangle */

        print("}")
    }
}
```

Fig. 12. Algorithm EX2XY



U.S. Coal Supply, Disposition, and Prices

Report No.: DOE/EIA 0584 (2007)
Report Released: September 2008
Next Release Date: September 2009

Table ES1. xls pdf Annual Coal Report

Table ES1. U.S. Coal Supply, Disposition, and Prices, 2006-2007
(Million Short Tons and Dollars per Short Ton)

Item	2006	2007
Production by Region		
Appalachian	391.2	377.8
Interior	151.4	146.7
Western	619.4	621.0
Refuse Recovery	0.8	1.2
Total	1,162.8	1,146.6
Consumption by Sector		
Electric Power	1,026.6	1,045.1
Coke Plants	23.0	22.7
Other Industrial Plants	59.5	56.6
Residential/Commercial	3.2	3.5
Total	1,112.3	1,128.0

(a)

Table ES1. U.S. Coal Supply, Disposition, and Prices, 2006-2007
(Million Short Tons and Dollars per Short Ton)

Item	2006	2007
Production by Region		
Appalachian	391.2	377.8
Interior	151.4	146.7
Western	619.4	621.0
Refuse Recovery		
Total		
Consumption by Sector		
Electric Power		
Coke Plants		
Other Industrial Plants		
Residential/Commercial		
Total		
Year-End Coal Stocks		
Electric Power		

(b)

Table ES1. U.S. Coal Supply, Disposition, and Prices, 2006-2007
(Million Short Tons and Dollars per Short Ton)

			2006	2007
Item	Production by Region	Appalachian	391.2	377.8
		Interior	151.4	146.7
		Western	619.4	621.0
		Refuse Recovery	0.8	1.2
		Total	1,162.8	1,146.6
	Consumption by Sector	Electric Power	1,026.6	1,045.1
		Coke Plants	23.0	22.7
		Other Industrial Plants	59.5	56.6
		Residential/Commercial	3.2	3.5
		Total	1,112.3	1,128.0
		Year-End Coal Stocks		
		Electric Power	141.0	151.2
		Coke Plants	2.9	1.9

(c)

Fig. 13. Part of a US Energy Information Administration table. (a) As it appears on the web; (b) Imported into Excel; (c) After editing. http://www.eia.doe.gov/cneaf/coal/page/acr/tables1.html

3 A Grammar for Table Candidates

Although EX2XY produces X-Y trees as verbose XML files, in this section we represent the trees with nested parentheses notation. This notation has a 1:1 correspondence with general trees provided that the order of the symbols is preserved [27]. We present the notation and the *Look Ahead Left to Right* (LALR) grammar $\mathfrak{G}_1$ [28,29] constructed to parse the X-Y trees of table-like tessellations by means of an example. The grammar was implemented in yacc [30].

Consider the following column headings for two Wang categories of Employment Status and Education (Fig. 14) which result in the derivation of Fig. 15.

Employment Status					
Unemployed			Employed		
Education					
High School or Less	College		High School or Less	College	
	BS/BA	Graduate Degree		BS/BA	Graduate Degree

Fig. 14. Sample table row heading for grammar $\mathfrak{G}_1$

Textual labels (like Employment Status) have no bearing on the structure, so we will replace them by the generic symbol c. We alternate brackets and braces for ease of reading, but they are equivalent. The X-Y tree "sentence" S_{XY} for this partition of the tessellation is:

$$S_{XY} = \{ c [c c] c [c \{ c [c c] \} c \{ c [c c] \}] \}$$

Grammar $\mathfrak{G}_1$ for parsing all layout-equivalent tessellations of this kind is:

```
S := A
A := {B}
B := c[X]B  |  c[X]
X := cX  |  AX  |  A  |  c
```

This grammar can parse fully parenthesized input for column headers of tables with arbitrary dimensions and any number of levels in each dimension. It is a simple matter to add a mirror-image grammar to parse the row headings and delta cells. The non-terminals in $\mathfrak{G}_1$ serve the following functions.

S is the start symbol (eventually to generate all admissible strings for tables).
A is the nonterminal that generates all admissible strings for column headers.
B generates one or more instances of categories in the form "c[X]".

Each c becomes a root category and X generates its subcategory tree. X generates strings of length ≥ 1, with arbitrary occurrences of c and A.

We rewrite the grammar in the following equivalent form for ease of reference:

$\mathbb{G}_1$ RULES:

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

Action	Stack state	Remaining Input
-	Null	{ c [c c] c [c { c [c c] } c { c [c c] }] }
Shift^5	{ c [c **c**	] c [c { c [c c] } c { c [c c] }] }
R 8	{ c [**c X**	] c [c { c [c c] } c { c [c c] }] }
R 5	{ c [X	] c [c { c [c c] } c { c [c c] }] }
Shift^9	{ c [X] c [c { c [c **c**	] } c { c [c c] }] }
R 8	{ c [X] c [c { c [**c X**	] } c { c [c c] }] }
R 5	{ c [X] c [c { c [X	] } c { c [c c] }] }
Shift	{ c [X] c [c { **c [X]**	} c { c [c c] }] }
R 4	{ c [X] c [c { B	} c { c [c c] }] }
Shift	{ c [X] c [c **{ B }**	c { c [c c] }] }
R 2	{ c [X] c [c A	c { c [c c] }] }
Shift^6	{ c [X] c [c A c { c [c **c**	] }] }
R 8	{ c [X] c [c A c { c [**c X**	] }] }
R 5	{ c [X] c [c A c { c [X	] }] }
Shift	{ c [X] c [c A c { **c [X]**	}] }
R 4	{ c [X] c [c A c { B	}] }
Shift	{ c [X] c [c A c **{ B }**	] }
R 2	{ c [X] c [c A c **A**	] }
R 7	{ c [X] c [c A **c X**	] }
R 5	{ c [X] c [c **A X**	] }
R 6	{ c [X] c [**c X**	] }
R 5	{ c [X] c [X	] }
Shift	{ c [X] **c [X]**	}
R 4	{ **c [X] B**	}
R 3	{ B	}
Shift	**{ B }**	
R 2	**A**	
R 1	S	

Fig. 15. Derivation for the example of Fig. 14

An LALR is a shift-reduce parser that at each step either *shifts* the next input on to the stack or *reduces* the symbols on top of the stack according to a rule of the grammar. It produces leftmost reductions as it scans the input from left to right, which yields a rightmost derivation in reverse order. The first column in Fig. 15 shows the action (shift or reduce); with Shift^n denoting n consecutive shifts and $\mathsf{R}\ m$ denoting reduction according to rule number m. The bold characters represent the *handle* (right-hand side) of the production that is reduced to the left-hand symbol by the rule listed on the next row.

This example demonstrates both the power and the limitations of using a grammatical approach to parsing: A grammar can be written to recognize a broad class of tilings. On the other hand, a context-free grammar is not powerful enough to check that the headings are labeled appropriately for a WFT. If a candidate structure is accepted by $\mathfrak{G}_1$, then we must conduct additional geometrical alignment and lexical checks to verify the Wang Notation.

4 X-Y Tree to Wang Notation

In this section we demonstrate XY2WANG an algorithm that converts an X-Y tree generated from a restricted family of *admissible tables* to Wang Notation. An example of this family can be seen in Figure 4(a). Figure 16 shows a simple example from this family that that XY2WANG can process. Although in Section 3 we used parenthesis notation for the trees, here we use an indented table-of-contents that reveals the underlying data structure (Figure 17). Figure 18 shows the top level pseudo-code for XY2WANG (which was implemented in Python and produces XML output). The algorithm accepts table trees with an arbitrary number of categories and levels of headings. For the selected example, the algorithm returns the Wang notation (in a verbose XML format) for a two-category table $T = (C,d)$:

Category Notation (Labeled Domains):

$$C = \{ \ (A,\{ (A_1,F),(A_2,\Phi) \}), (B,\{ (B_1,\Phi),(B_2,\Phi),(B_3,\Phi) \}) \ \}$$

Delta Mappings:

$$\delta(\{A.A_1,B.B_1\}) = d_{11}$$
$$\delta(\{A.A_1,B.B_2\}) = d_{12}$$
$$\delta(\{A.A_1,B.B_3\}) = d_{13}$$
...

The algorithm first locates the four principal regions of the table in the XY tree: the stub, row-headings, column-headings, and content cells. It then extracts the Wang labeled domains from the category regions under the assumption that each spanning cells in the row header is the parent category of smaller cells to its right, and each spanning cell in the column header is the parent of smaller cells below it. After the category notation is generated, the Cartesian product of the category paths is computed and each key is matched to the content of a delta cell.

		B		
		B1	B2	B3
A	A1	d11	d12	d13
	A2	d21	d22	d23

Fig. 16. Example table for XY2WANG

Index	Node	Parent	Children	H x W
1	Outer Frame	None	2,9	4x5
2	Left Side	1	3,4	2x5
3	Stub	2	None	2x2
4	A+(A1+A2)	2	5,6	2x2
5	A	4	None	2x1
6	(A1+A2)	4	7,8	2x1
7	A1	6	None	1x1
8	A2	6	None	1x1
9	Right Side	1	10,11,15,19	4x3
10	B	9	None	1x3
11	B1+B2+B3	9	12,13,14	1x3
12	B1	11	None	1x1
13	B2	11	None	1x1
14	B3	11	None	1x1
15	d11+d12+d13	9	16,17,18	1x3
16	d11	15	None	1x1
17	d12	15	None	1x1
18	d13	15	None	1x1
19	d21+d22+d23	9	20,21,22	1x3
20	d21	19	None	1x1
21	d22	19	None	1x1
22	d23	19	None	1x1

Fig. 17. Data structure created by XY2WANG for the X-Y tree of the table in Fig. 17. The index represents a depth-first traversal of the X-Y tree, which has 8 internal nodes (including the root) and 14 leaf nodes corresponding to the cells of the table. Children are listed top-to-bottom or left-to-right. Borders enclose the four principal regions of the table.

XY2WANG must be able to handle more complex scenarios than Figure 16, such as higher Wang dimensionality, deeper nesting of headers, repetitive headers, and the detection of not well-formed tables. Provisions for such scenarios are included in the Python program outlined by the pseudo-code of Figure 18.

```
Pseudo-code for XY2WANG

Divide Left Side into Stub and Row-headings // (Stub is first child of Left Side)
Divide Right Side into Col-headings and Data Cells // separate using stub-height
Separate row-category subtrees at nodes with spanning heights
Separate column-category subtrees at nodes with spanning widths
Traverse breadth-first each category tree while removing duplicate labels
Return Wang Category notation
Form Cartesian product of unique paths
Compare size of product to number of data cells
If it differs from number of leaf nodes in Data Cells, return “tree not well formed”
Else assign a data cell in Data Cells to each path
Return Wang Delta notation
```

Fig. 18. Top-level pseudo-code for algorithm XY2WANG

5 Conclusion

Web tables intended for human readers are generally laid out on a grid. The data cells are referenced by row and column headings which form labeled domains of categories. The hierarchical structure of categories and the flat structure of the data cells can be recovered by interleaved vertical and horizontal partitions represented as X-Y trees. An X-Y tree represents a generic rectangular tiling and indiscriminately makes all the cuts in each direction. The table grammar reorders the cuts so as to represent the structure of the table according to specific style(s) of tables.

We defined geometric and topological equivalence classes on tessellations and their X-Y trees. Many tables encountered in practice correspond to well-defined subsets of these equivalence classes. They can be identified by parsing the X-Y tree with a context-free grammar. If the labels of the headings are consistent, then the table is well formed, and we can algorithmically extract its Wang category notation.

The current formalism does not account for *augmentations* although our experimental system does process them and includes them in the XML output. Common augmentations are *aggregates* (sums, averages and weighted averages, medians), *footnotes, units, annotations, table titles* and *captions*. As Wang noted and our 200-table experiment confirms, these are essential components of most tables. Because they are not revealed by the tiling itself, so far we have not been able to treat them uniformly, but they must eventually be integrated into any practical table understanding system.

The precise representation of layout-invariant table syntax is the first step towards semantic interpretation of groups of conceptually overlapping tables. The approach we propose towards this goal is to import the web tables into a spreadsheet, interactively edit them as necessary, and then algorithmically transform the data into Wang Notation in a portable XML format. We believe that syntactic analysis of the X-Y trees will allow identifying tables requiring similar edit steps, so that these edit steps can be applied automatically. This will effectively expand the number of admissible layouts and thereby reduce the amount of necessary interaction.

Acknowledgments. This work was supported by the National Science Foundation under Grants# 044114854 and 0414644 and by the Rensselaer Center for Open Source Software. We gratefully acknowledge the influence of two decades of discussions about tables and ontologies with Prof. D.W. Embley of Brigham Young University.

References

1. Tijerino, Y.A., Embley, D.W., Lonsdale, D.W., Nagy, G.: Towards ontology generation from tables. World Wide Web Journal 6, #3 (2005)
2. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (2001)
3. Halevy, A., Norvig, P., Pereira, F.: The Unreasonable Effectiveness of Data. IEEE Transactions on Intelligent Systems, 8–12 (March/April 2009)
4. Lopresti, D., Embley, D.W., Hurst, M., Nagy, G.: Table Processing Paradigms: A Research Survey. Int. J. Doc. Anal. Recognit. 8(2-3), 66–86 (2006)
5. Zanibbi, R., Blostein, D., Cordy, J.R.: A survey of table recognition: Models, observations, transformations, and inferences. Int. J. Doc. Anal. Recognit. 7(1), 1–16 (2004)
6. Silva, E.C., Jorge, A.M., Torgo, L.: Design of an end-to-end method to extract information from tables. Int. J. Doc. Anal. Recognit. 8(2), 144–171 (2006)
7. Gatterbauer, W., Bohunsky, P., Herzog, K.M., Pollak, B.: Towards Domain-Independent Information Extraction from Web Tables. In: Proceedings of World Wide Web, Banff, pp. 71–80 (2007)
8. Padmanabhan, R., Jandhyala, R.C., Krishnamoorhty, M., Nagy, G., Seth, S., Silversmith, W.: How many different kinds of tables are there. In: Procs. Eights Int'l. Workshop on Graphics Recognition (GREC 2009) (2009) (in press)
9. Klarner, D.A., Magliveras, S.S.: Tilings of a Block with Blocks. Europ. J. Combinatorics 9, 317–330 (1988)
10. Kuh, E.S., Ohtsuki, T.: Recent Advances in VLSI Layout. Proceedings of the IEEE 78(2) (1990)
11. Brooks, R.L., Smith, C.A.B., Stone, A.H., Tutte, W.T.: The dissection of rectangles into squares. Duke Math. J. 7, 312–340 (1940)
12. Nagy, G., Seth, S.: Hierarchical Image Representation with Application to Optically Scanned Documents. In: Procs. Int. Conf. Pat. Recog. VII, Montreal, pp. 347–349 (1984)
13. Krishnamoorthy, M., Nagy, G., Seth, S., Viswanathan, M.: Syntactic Segmentation and Labeling of Digitized Pages from Technical Journals. IEEE Transactions on Pattern Analysis and Machine Intelligence 15, #7, 737–747 (1993)
14. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, San Francisco (2006)
15. The Chicago Manual of Style, 15th edn. Univ. of Chicago Press, Chicago (2003)
16. U.S. Government Style Manual, 29th edn. (2000)
17. Green, E.A., Krishnamoorthy, M.: Model-based analysis of printed tables. In: Procs. of Third International Conference on Document Analysis and Recognition (ICDAR 1995), Montreal, Canada, pp. 214–217 (1995)
18. Green, E.A., Krishnamoorthy, M.: Recognition of tables using table grammars. In: Procs. of Symposium on Document Analysis and Recognition (SDAIR 1995), Las Vegas, NV, pp. 261–277 (1995)

19. Green, E.A., Krishnamoorthy, M.: Model-based analysis of printed tables. In: Procs. Third Int'l. Workshop on Graphics Recognition (GREC 1995), pp. 234–242 (1995); in Graphics Recognition Methods and Applications. LNCS, vol. 1072, pp. 80–91. Springer, Heidelberg (1996)
20. Wang, X.: Tabular Abstraction, Editing, and Formatting. Ph.D Dissertation, University of Waterloo, Waterloo, ON, Canada (1996)
21. Hu, J., Kashi, R., Lopresti, D., Nagy, G., Wilfong, G.: Why table ground-truthing is hard. In: Procs. of Sixth International Conference on Document Analysis and Recognition, Seattle, WA, pp. 129–133 (2001)
22. Lopresti, D., Nagy, G.: A Tabular Survey of Table Processing. In: Chhabra, A.K., Dori, D. (eds.) GREC 1999. LNCS, vol. 1941, pp. 93–120. Springer, Heidelberg (2000)
23. Nagy, G., Lopresti, D.: Issues in ground-truthing graphic documents. In: Blostein, D., Kwon, Y.-B. (eds.) GREC 2001. LNCS, vol. 2390, pp. 46–66. Springer, Heidelberg (2002) (selected papers from the Fourth International Workshop on Graphics Recognition)
24. Tao, C., Embley, D.W.: Automatic Hidden-Web Table Interpretation by Sibling Page Comparison. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 566–581. Springer, Heidelberg (2007)
25. Tao, C., Embley, D.W., Liddle, S.W.: Enabling a Web of Knowledge. Brigham Young University. manuscript submitted to the special issue about the web of data for the Journal of Web Semantics (2009)
26. Embley, D.W., Lopresti, D., Nagy, G.: Notes on Contemporary Table Recognition. In: Bunke, H., Spitz, A.L. (eds.) DAS 2006. LNCS, vol. 3872, pp. 164–175. Springer, Heidelberg (2006)
27. Horowitz, E., Sahni, S.: Fundamentals of Data Structures. W.H. Freeman & Co., New York (1983)
28. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools. Addison-Wesley, Reading (1986)
29. DeRemer, F., Pennello, T.: Efficient Computation of LALR(1) Look-Ahead Sets. ACM Trans. Prog. Lang. and Sys. (TOPLAS) 4(4), 615–649 (1982)
30. Johnson, S.C.: YACC: Yet another Compiler-Compiler. Unix Programmer's Manual 2b (1979)