2-11-2016

Converting heterogeneous statistical tables on the web to searchable databases

David W. Embley
Brigham Young University, embley@cs.byu.edu

Mukkai S. Krishnamoorthy
Rensselaer Polytechnic Institute, moorthy@cs.rpi.edu

George Nagy
Rensselaer Polytechnic Institute, nagy@ecse.rpi.edu

Sharad C. Seth
University of Nebraska - Lincoln, seth@cse.unl.edu

Published in *International Journal on Document Analysis and Recognition (IJDAR)* 19:2 (June 2016), pp 119–138.
doi 10.1007/s10032-016-0259-1
Submitted 24 April 2015; revised 21 January 2016; accepted 25 January 2016; published 11 February 2016.

Converting heterogeneous statistical tables on the web to searchable databases

David W. Embley,[1] Mukkai S. Krishnamoorthy,[2] George Nagy,[2] and Sharad Seth[3]

1 Computer Science Department, Brigham Young University, Provo, UT 84602
2 Rensselaer Polytechnic Institute, Troy, NY 12180
3 University of Nebraska–Lincoln, Lincoln, NE 68588

Correspondence — David W. Embley embley@cs.byu.edu
Mukkai S. Krishnamoorthy moorthy@cs.rpi.edu
George Nagy nagy@ecse.rpi.edu
Sharad Seth seth@cse.unl.edu

Abstract
Much of the world's quantitative data reside in scattered web tables. For a meaningful role in Big Data analytics, the facts reported in these tables must be brought into a uniform framework. Based on a formalization of header-indexed tables, we proffer an algorithmic solution to end-to-end table processing for a large class of human-readable tables. The proposed algorithms transform header-indexed tables to a category table format that maps easily to a variety of industry-standard data stores for query processing. The algorithms segment table regions based on the unique indexing of the data region by header paths, classify table cells, and factor header category structures of two-dimensional as well as the less common multi-dimensional tables. Experimental evaluations substantiate the algorithmic approach to processing heterogeneous tables. As demonstrable results, the algorithms generate queryable relational database tables and semantic-web triple stores. Application of our algorithms to 400 web tables randomly selected from diverse sources shows that the algorithmic solution automates end-to-end table processing.

Keywords Document analysis, Table segmentation, Table analysis, Table header factoring, End-to-end table processing, Table headers, Queries over table data

1 Introduction

Tables provide a convenient and succinct way to communicate data of interest to human readers. Cafarella and others called attention to the immense accumulation of tabulated data on the web even before Big Data became a byword [1]. Assuming "that an average table contains on average 50 facts it is possible to extract more than 600 billion facts taking into account only the 12 billion sample tables found in the Common Crawl" [2].

Tables are not, however, inherently amenable to machine-based search and query. Research on document image analysis suggests that there is a natural progression from source document images to a searchable database via "physical" and "logical" layout analysis. In the case of tables, physical analysis must assign literal content to cells laid out on a grid. Logical analysis determines the indexing relationship between header cells and data cells. The indexing structure can be readily converted to any appropriate machine-queryable representation such as relations in a relational database or subject-predicate-object fact assertions in a semantic-web triple store. We propose here a complete and coherent table-processing framework to accomplish all of these tasks. We call the constraints necessary to solve the ill-posed inverse problem of table understanding *table regularization*. The exemplary table in Fig. 1 will serve to illustrate the analysis of physical and logical layout and the assertion of facts in machine-queryable form. Although our methods could be applied to scanned tables, here we address only tables where the basic grid structure and the cell contents are already available in encoded form.

- *Physical layout* All tables have a grid structure. Every literal (word, phrase, or numerical value) has a row and a column coordinate. In Fig. 1, as in most tables, the data values form a natural grid. When spanning header labels (*Country*, *Million dollar*, and *Percentage of GNI* in Fig. 1) are replicated into the cells they span, the header labels also become part of the grid. Because we also process table titles, footnotes, and other notes associated with tables, we treat these auxiliary components as spanning cells and replicate them across the row (or column) of grid cells in which they appear. Our processing chain starts with a grid, as described here, because HTML and spreadsheet tables are already built on a grid. As shown below, methods have been developed earlier for converting scanned, ASCII, and searchable PDF tables to a grid of cells in spite of the variety of framing, partial ruling, typeface, color scheme, and cell formatting details. Explicit distinctions between cells containing table title, data values, row and column headers, and footnotes, however, are totally absent in our initial grid representation. Furthermore, there are no rulings that might indicate divisions between data values and other parts of a table, and cell content is just text without color or font formatting.

Statistics Norway
Statistisk sentralbyrå

Expenditures on development aid in the OECD countries

1 Official development assistance.

Country	Million dollar					Percentage of GNI				
	2007	2008	2009	2010*	2011*	2007	2008	2009	2010*	2011*
Norway	3 735	4 006	4 081	4 580	4 936	0.95	0.89	1.06	1.10	1.00
Denmark	2 562	2 803	2 810	2 871	2 981	0.81	0.82	0.88	0.91	0.86
Finland	981	1 166	1 290	1 333	1 409	0.39	0.44	0.54	0.55	0.52
Sweden	4 339	4 732	4 548	4 533	5 606	0.93	0.98	1.12	0.97	1.02
Belgium	1 951	2 386	2 610	3 004	2 800	0.43	0.48	0.55	0.64	0.53
France	9 884	10 908	12 602	12 915	12 994	0.38	0.39	0.47	0.50	0.46
Greece	501	703	607	508	331	0.16	0.21	0.19	0.17	0.11
Ireland	1 192	1 328	1 006	895	904	0.55	0.59	0.54	0.52	0.52
Italy	3 971	4 861	3 297	2 996	4 241	0.19	0.22	0.16	0.15	0.19
Luxembourg	376	415	415	403	413	0.92	0.97	1.04	1.05	0.99
Netherlands	6 224	6 993	6 426	6 357	6 324	0.81	0.80	0.82	0.81	0.75
Portugal	471	620	513	649	669	0.22	0.27	0.23	0.29	0.29
Spain	5 140	6 867	6 584	5 949	4 264	0.37	0.45	0.46	0.43	0.29
United Kingdom	9 849	11 500	11 283	13 053	13 739	0.36	0.43	0.51	0.57	0.56
Switzerland	1 685	2 038	2 310	2 300	3 086	0.38	0.44	0.45	0.40	0.46
Germany	12 291	13 981	12 079	12 985	14 533	0.37	0.38	0.35	0.39	0.40
Austria	1 808	1 714	1 142	1 208	1 107	0.50	0.43	0.30	0.32	0.27
Canada	4 080	4 795	4 000	5 209	5 291	0.29	0.33	0.30	0.34	0.31
United States	21 787	26 437	28 831	30 353	30 745	0.16	0.18	0.21	0.21	0.20
Japan	7 697	9 601	9 457	11 021	10 604	0.17	0.19	0.18	0.20	0.18
South Korea	696	802	816	1 174	1 321	0.07	0.09	0.10	0.12	0.12
Australia	2 669	2 954	2 762	3 826	4 799	0.32	0.32	0.29	0.32	0.35
New Zealand	320	348	309	342	429	0.27	0.30	0.28	0.26	0.28
OECD/DAC[1] countries total	**104 206**	**121 954**	**119 778**	**128 465**	**133 526**	**0.27**	**0.30**	**0.31**	**0.32**	**0.31**

1 DAC-countries are members of OECD's Development Assistance Committee.

Source: OECD.

Explanation of symbols

Fig. 1. A table from Statistics Norway, used as a running example throughout the paper. http://www.ssb.no/a/english/kortnavn/uhjelpoecd_en/tab-2012-05-15-01-en.html (Accessed Jan 2015).

Surprisingly, this lossy representation of an original table often suffices to automatically extract the fact assertions stated therein.

- *Logical layout* Starting with a table as a grid of text-filled or empty cells, we reveal its indexing structure in terms of categories and an ordered list of category paths for each data cell. The table in Fig. 1 has three hierarchical header categories: *Country* (*Norway, Denmark, ...*), *Year (2007, 2008, ...*), and *development assistance* (*Million dollar, Percentage of GNI*). The index for each data value comprises one header path from each category tree. The upper-left data value 3 735 in the table, for example, is indexed by: (*Country.Norway, Year.2007, development_ assistance. Million_dollar*). This representation mirrors Wang's formalization of indexing in tables [3], which maps a 2-D grid table into an *n*-D array with coordinates corresponding to the categories, i.e., a data cube.

- *Fact assertions* The final output of our table-processing work is a collection of fact assertions, represented as relational database tables and also as subject-predicate-object triples in a semantic-web standard. Each data value in a table makes a fact assertion. The assertion for the data value *3 735* in Fig. 1, is: *The Country Norway in Year 2007 provided development assistance in the amount of 3 735 Million dollars*. Our table-processing system yields these assertions in a form that can be queried with standard query languages—SQL for relational database tables and SPARQL for semantic-web triples. When table headers agree, cross-table query processing is possible, as illustrated in Sect. 7.We also identify auxiliary information, comprising titles, footnotes, footnote markers and references, and notes, and turn their existence into fact assertions, which can then be queried as such.

Whereas most previous work addresses specific types of tables, we exploit the commonality of the grid format and indexing structure. Human readers often depend on rulings, fonts, and typesetting to reveal the intrinsic relationship between headers and content cells, but our method relies only on structural constraints. We also extract embedded auxiliary data without dependence on formatting.

We do not deal herewith concatenated (composite) tables, nested tables, tables containing graphic data, or "egregious" tables (those not laid out on a rectangular grid with headers above and left).

Although most research on document processing is experimental, our table-processing work makes several theoretical contributions that have immediate practical applications. We provide

1. a formal (block grammar) definition of header-indexed tables that can be used for analysis of most human-readable tables;
2. an automatic transformation of header-indexed tables to a new canonical category table format via:
   (a) segmenting table regions by algorithmic data cell indexing,
   (b) factoring header paths into categories by algorithmic header analysis, and
   (c) generating queryable canonical relational tables and semantic-web triple stores.

Our program accepts rectangular tables posted on the web for human reading in HTML XLS or CSV format. Some publishers already include CSV tables in online versions of published papers. The input tables are *heterogeneous* in the sense that they are not restricted to any specific domain or by any formatting constraint. Their headers could have any reasonable number

of rows or columns. Multiple header hierarchies could be indicated by any combination of spanning cells. The tables could have footnotes, footnote references, or other notes. They are just web tables, generated either manually or from some database, posted for human reading. Our program always finds a row header, a column header, and an indexing structure. These do not, however, necessarily correspond to what a sensible human may have assigned as ground truth. For example, a row of units may be assigned to the data region rather than to notes. In principle, the input tables could have been produced by any of the earlier methods for transforming scanned tables into computer-readable grid tables, but we have not yet experimented with scanned tables. Although our test data consist of tables from statistical sites, we have carefully avoided dependence on statistical or numerical data.

We find it remarkable that random collections of heterogeneous tables can be segmented by reliance on the indexing property of their row and column headers.

After reviewing relevant prior research in Sect. 2, we present in Sect. 3 classical (printing and publishing) table terminology and formalize header-indexed tables in terms of a block grammar. We explain how our table-processing software segments and classifies cells in Sect. 4 and how it finds categories, assigns indexes for data cells, and produces category tables in Sect. 5. In Sect. 6, we validate our work over a collection of tables. Section 7 shows SQL and SPARQL queries to demonstrate that the human-readable tables are indeed converted into data stores of machine-queryable fact assertions. In Sect. 8, we draw conclusions and point to further research opportunities.

2 Prior work

Ulpian's life-expectancy tables [4] indicate that presenting related data in rows and columns was already familiar to the Romans, but systematic use of scientific tables did not come about until the seventeenth century. Over the last 40 years, the prospect of computer access to data available in tables stimulated several hundred research projects on table analysis. Diverse methods were developed for bitmapped images of scanned or digitally photographed hardcopy tables, ASCII tables found in email messages or in early computer-generated documents, searchable or raw PDF files, and both manually coded and automatically generated spreadsheet and HTML tables. We describe previous table models and summarize published methods of table analysis (variously called *table recognition, table interpretation, table understanding,* or *table data extraction*).

This literature review has four parts. We first review X. Wang's pioneering research which has long guided our approach to table understanding.

In the second subsection we point out research that justifies our claim that table spotting, table isolation, and conversion of source tables to grid tables is no longer major obstacles to table understanding. Next we review research that aims, like ours, at higher-level, logical analysis of tables. Finally, we summarize our own previous work that underlies our current endeavors. For a thorough survey of earlier work, we recommend [5].

2.1 Wang tables

Wang regarded tables as an abstract data type [3]. She formalized the distinction between physical and logical structure in the course of building X-Table for practical table composition in a Unix X-Windows environment. She defined *layout structure* as the presentation form of a table and *logical structure* as a set of labels and values. Labels are assigned to hierarchies of categories and subcategories, and each value in a data cell is associated with one label from each of the categories. The number of categories defines the dimensionality of the abstract table.

More specifically, Wang formulated the logical structure of a table in terms of *category trees* corresponding to the header structure of the table [3]. "Wang categories," a form of multidimensional indexing, are defined implicitly by the 2-D geometric indexing of the data cells by row and column headers. The index of each data cell is unique (but it may be multidimensional and hierarchical in spite of the flat, two-dimensional physical layout of the table). She used the object-oriented dot notation, *label*1.*label*2.*label*3, to represent a path in the category tree from header cells to data cells. Thus, for example, Wang would identify the three category trees in Fig. 1 for countries, years, and development assistance, and index each data cell as a triple of paths, one for each category tree.

2.2 Physical structure extraction (low-level table processing)

In printed tables, boxing, rules, or white-space alignment is used for separating cell entries. In one of the earliest works, Laurentini and Viada extracted cell corner coordinates from the ruling lines [6]. Image processing techniques for the extraction of physical structure from scanned tables include Hough transforms [7], run-length encoding [8],word bounding boxes [9], and conditional random fields (CRF) [10]. Hirayama presented an algorithm for segmenting partially ruled tables into a rectangular lattice [11]. Handley's method of iterative identification of cell separators successfully processed large, complex, fully lined, semi-lined, and unruled tables with multiple lines of text per cell [12]. Zuyev used connected components and projection profiles to identify the cell contents for an OCR system [13]. Methods for detecting and locating tables were demonstrated in [14] and [15].

The notion of converting paper tables into Excel spreadsheets dates back at least to 1998 [16]. Early research on table processing suffered from the isolation of the graphics research community from the OCR community. Current OCR products can locate tables on a printed page and convert them into a designated (e.g., word-processor) table format. Most desktop publishing software has provisions for the interconversion of tables and spreadsheets. Our methods are applicable to scanned tables segmented as prescribed in [6– 8,10–12], provided that cell contents are converted to ASCII even with mediocre OCR. Related research addressing raw PDF tables, which requires recovering the grid structure as well as OCR for the label contents, was recently presented in [17].

Less attention has been focused on ASCII table analysis, where the structure must often be discovered from the correlation of text blocks on successive lines. Grid structure is preserved by spacing, although vertical separators ("|") and extra new-line symbols for blank rows or rows filled with dashes are sometimes used. Pyreddy and Croft demonstrated results on over 6000 tables from the Wall Street Journal [18]. T-Recs clustered words for bottom-up structural analysis of ASCII tables [19]. Hu et al. explored row and column alignment via directed acyclic attribute graphs [20].Work on such tables has diminished since the development of XML for communicating structured data without sacrificing ASCII encoding.

Figure 2a shows some of the cells in the exemplary table and the HTML tags that preserve table topology. The tagging makes the extraction of a table's underlying grid structure from its customary HTML representation relatively simple. Figure 2b shows the limited information retained when the HTML representation in Fig. 2a is converted into CSV format. In the CSV file (1) the labels of spanning cells are followed by delimiters (here commas) that form a full grid of cells; and (2) all type and cell formatting and ruling lines are removed. Excel displays files with an equal number of delimiters between new-line symbols as a table. Excel does not retain appearance-based edits when the file is saved in CSV format.

2.3 Logical structure extraction (high-level table processing)

Gattebauer et al. presented a geometric approach to table extraction from arbitrary web pages based on the spatial location of table elements prescribed by the DOM tree [23]. They formulated a "visual table-model" of nested rectangular boxes derived from Cascading Style Sheets. They applied spatial reasoning—primarily based on adjacency topology and Allen interval relations—to their visualization model in order to determine the final box structure, and conducted some semantic analysis with a known or assumed list of keywords. Their interpretation consists of XML-tagged generalized

```
<html>
...
<!-- START TABELL -->
...
<tr>
<throwspan=2class=level11>Country</th>
<thclass=multispancolspan=5style=border-bottom:
1px #000000 solid; >Million dollar</th>
<thclass=multispancolspan=5style=border-bottom:
 1px #000000 solid; >Percentage of GNI</th>
</tr>
...
<tr>
<th>2007</th>
<th>2008</th>
<th>2009</th>
<th>2010*</th>
...
</tr>

<!-- SLUTT TABELL -->
...
</html>
```

(a)

```
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
Country,Million dollar,,,,,Percentage of
GNI,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,2007,2008,2009,2010*,2011*,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
```

(b)

Fig. 2. File representation of tables. We import HTML [21] or XLSX files and convert them into CSV [22] files that preserve only the grid structure and labels without font type, size, color, and spacing. **(a)** Some of the 446 line source code of the HTML table in Fig. 1. **(b)** Text (Notepad) display of the same part of the CSV file after import from the HTML in **(a)**.

n-tuples. They evaluated several steps of their process on 269 web pages with 493 tables and reported 48% precision and 57% recall.

Amano and Asada have published a series of papers on graph grammars based on box adjacency for "table-form" documents [24]. Their grammars encode the relationship between "indicator," "example," and data boxes. Similarities between table and form processing were already emphasized by Bing et al. [25] and Kieninger and Dengel [26]. Grammar-based approaches

that can be specialized to forms and tables have been demonstrated on large data sets [27–29]. However, bureaucratic forms often have preprinted labels rather than indexing headers like tables. Forms like invoices are less tightly structured than tables [30]. Therefore we cannot take advantage of advanced forms processing methods like [31,32].

A group headed by T. Watanabe aimed at learning the various types of information necessary to interpret a ruled scanned table [33]. They used a training set of diverse tables to populate a "Classification Tree." The nodes of the tree are "Structure Description Trees" that can interpret a specific family of tables. In their operational phase, new classification nodes and tree structure descriptions are added for unrecognized tables.

Shamalian et al. demonstrated a model-based table reader for reading batches of similar tables [34]. Their model specifies the location of the data cells, thus obviating the need to interpret headers either syntactically or semantically.

Table headers in PDF files were detected and analyzed in [35] in order to classify table types. A rule-based system with goals similar to ours was presented in [36].

In the last several years, an active and inventive group at Google, possibly inspired by Halevy, Norvig, and Pereira [37], collected and analyzed millions of tables harvested from the web [1,38,39]. Visual verification of their results has necessarily been restricted to much smaller samples. Their general approach has been to treat table rows as tuples with attributes specified by the top row. Extending this work to tables more complex than simple relational tables, Adelfio and Samet leveraged the principles of table construction to generate interpretations for spreadsheet and HTML tables [40]. Using Conditional Random Fields, they classified each table row as: *header*, *data*, *title*, *group header*, *aggregate*, *non-relational metadata*, or *blank*. With their test set of 1048 spreadsheet tables and 928 HTML tables, they achieved an accuracy of 76.0% for classifying header and data rows for spreadsheet tables and 85.3% for HTML tables, and for classifying all rows, 56.3 and 84.6%, respectively. In contrast to the work of the Google group and of Adelfio and Samet, we treat row headers the same as column headers, and instead of depending on appearance features, we use indexing properties for further analysis.

A series of papers culminating in V. Long's doctoral thesis [41] analyzes a large sample of tables from Australian Stock Exchange financial reports. An interesting aspect of this work is the detection and verification of the scope and value of *aggregates* like totals, subtotals, and averages. The analysis is based on a blackboard framework with a set of cooperating agents. This dissertation has a good bibliography of table papers up to 2009. Other work dealing with aggregates in tables includes [42].

Already in 1997, Hurst and Douglas advocated converting tables into relational form: *Once the relational structure of the table is known it can be manipulated for many purposes* [43]. Hurst provided a taxonomy of category attributes in terms of *is-a, part-of, unit-is, quantity-is*. He pointed out that the physical structure of a table is somewhat analogous to syntax in linguistic objects. He also emphasized the necessity and role of natural language analysis for table understanding, including the syntax of within-cell strings [44]. Hurst's dissertation contains a wealth of interesting examples of tables [45].

Hurst's work was reviewed and augmented by Costa e Silva et al. [46], who analyzed prior work in detail in terms of contributions to the tasks of *table location*, *segmentation*, *functional analysis* (tagging cells as data or attribute), *structural analysis* (header index identification), and *interpretation* (semantics). Costa e Silva's research group also provides a clear distinction between tables, forms, and lists. The ultimate objective of this group is the operational analysis of financial tables with feedback between the five tasks based on confidence levels.

Kim and Lee reviewed web table analysis from 2000 to 2006 and found logical hierarchies in HTML tables using cell formats and syntactic coherency [47]. They extracted the table caption and divided spanning cells correctly. Like us, but in contrast to many other researchers, they handled vertical and horizontal column headers symmetrically.

The TARTAR (Transforming ARbitrary TAbles into fRames) system developed by Pivk et al. has objectives similar to ours: "The input to the system is semi-structured information in the form of *arbitrary* (HTML, PDF, EXCEL, etc.) tables." [48]. However, in the cited paper, the authors demonstrated their work only on HTML tables. Their "cleaned and canonicalized" matrix representation is similar to our grid table. Downstream analysis and region segmentation proceeded, however, on the basis of cell formats (letters, numerals, capitalization, and punctuation) rather than indexing properties. The cells were functionally labeled in a manner similar to Hurst as *access* or *data* cells and assembled into a *Functional Table Model*. An attempt was made for semantic interpretation of strings using WordNet. The final output was a semantic (F-logic) frame. The complex evaluation scheme that was presented and applied to 158 HTML tables was hampered by human disagreement over the description of the frames.

Chen and Cafarella recently presented a table-processing system that transforms spreadsheet tables into relational database tables [49]. Like Adelfio and Samet [40] and Pinto et al. [10], they adapt a CRF to label each row as *title*, *header*, *data*, and *footnote*, using similar row features. (Rows labeled as "*data*" also include the cells in the row header, hence to distinguish between the two, they must assume, unlike us, that the data region is purely numeric.) Their hierarchy extractor builds parent-child candidates of cells in

the header region using formatting, syntactic, and layout features. The candidate list is pruned by an SVM classifier that enforces the resulting set of candidate pairs to be cycle-free. In our algorithmic approach to table processing, the resulting structure is guaranteed to be cycle-free by construction. Their corpus of tables was posted on-line, and we use a random sampling of these tables in our experiments.

Some researchers consider wholly automated table analysis too remote and advocate interactive methods based on expert advice and user feedback [50,51].

Our approach differs from previous work by its reliance on the fundamental indexing property of headers and by the completeness of its output in standard computer-searchable formats.

2.4 Our earlier work

We reviewed early work on table processing and presented a collection of tables that stretch the very definition of *table* in 1999 [52]. Examples of human ambiguity in table interpretation were discussed in [53]. The extent to which semantic information is revealed by table structure was explored in [54].We compiled a comprehensive survey of table processing for *IJDAR* in 2006 [55]. Input tables were matched with known conceptualizations in an attempt to interpret them in [56]. Information extraction from s*ibling tables* with identical headers was demonstrated in [57]. A taxonomy of tables based on the geometric relationship of tabular structures to isothetic tessellations and to X-Y trees was proposed in [58], a machine learning approach to segmentation of grid tables in [59], and algorithms for turning web tables into relational tables by recovering and factoring header paths in [60]. *VeriClick*, an interactive tool for table segmentation and ground-truthing, was described in [61]. We introduced algorithmic table segmentation, based on the fundamental indexing property, in [62]. Some other conference reports of our experiments on various aspects of table processing are cited in the above publications.

In addition to the already-mentioned IEA/AIE'11 [60] and ICDAR'13 [62] papers, three precursors to this article have recently appeared in conference proceedings. At the 2014 Document Analysis Systems workshop, we reported on our initial, automatic end-to-end conversion of web tables to relational databases [63]. We showed SQL queries on HTML tables imported into MS-Access at ICPR 2014 [64]. At the 2015 IST/SPIE Conference on Document Recognition and Retrieval, we clustered the headers of category hierarchies to reveal commonalities among tables [65].

The current paper combines and significantly expands these precursors. (1) The updated literature review contrasts prior work with ours. (2) We

describe header-indexed tables in terms of a block algebra that formalizes the conventional typesetting practices of the printing and publishing industry that underlie web tables [66]. (3) The MIPS (Minimum Indexing Point Search) and the category-tree extraction algorithm (i.e., header factoring) are reframed in terms of the new header-indexed table formalization. (4) Exercising these algorithms on a collection of heterogeneous tables, we present a detailed analysis of the required header modifications for Wang-category-tree construction. (5)We transform algorithmically discovered table content to semantic-web triple stores and to relational databases, and we execute both SQL and SPARQL queries over two hundred automatically processed HTML tables.

3 Human-readable tables

Good table layout is an art described in several books and in lengthy sections of the US Government Printing Office Style Manual and in the Chicago Manual of Style. In this section, we first informally present the generally accepted view of tables. We then specify a visual schematic model of the header-indexed tables that we can process. The model is formalized in a 2-D interval algebra over the inherent spatial constraints.

3.1 What is a table?

Tables are universally used for presenting data logically organized into two or more *categories*: *Country*, *Year*, and *development assistance* in Fig. 1. Their *data cells* are laid out on a grid so that each data cell can be indexed by its row and column headers. In conventional printing terminology, the *principal zone* of a table comprises regions called *stub head*, *row header* (or *stub*), *column header*, and *data*. Auxiliary information, such as the table title, notes, and footnotes appear outside this principal zone. Notes may also appear in the principal zone. The stub head may be empty or augment information carried by row or column headers, or the table title. In Fig. 1, the stub head contains *Country*.

A single category can be indexed by a flat header like the list of countries in Fig. 1, or by a hierarchical header laid out in several rows or columns or designated by indentations or font characteristics. Hierarchical headers also allow 2-D display of more than two categories by repeated labels.

Figure 1 displays two categories, *development assistance* and *Year* as hierarchies: *Million dollar (2007, ?, 2011*)* and *Percentage of GNI (2007, ?, 2011*)*.

Since horizontal and vertical table organization is symmetric and permutable, the number of possible table layouts increases combinatorially

with the number of categories and the number of their content labels. The choice may be guided by the aspect ratio of the available page or display space, preference for horizontal or vertical labels, compatibility with existing tables, and expected reader interests. Larger tables tend to be laid out with more rows than columns. Thus Canadian provinces often appear as column headers, while US states are typically row headers. The order of rows and columns does not affect indexing. When row order is significant, the leading column may be populated with integers denoting rank. Since these uniquely index all the remaining rows, they logically serve as row headers in spite of their descriptive poverty.

Every category is a *rooted tree*. Its root serves as its *Category Name*. In practice, it is often omitted because it is obvious to the reader. In Fig. 1, for example, the label *Year* does not appear (and could offend some readers if it did). Even when the category root is not missing, an arbitrary string (e.g., *RootHeader#2*) may be inserted to complete the category structure because category roots cannot affect indexing. Our algorithms always assign a virtual root because assigning a meaningful name could require semantic analysis of the contents of the table, table title, notes, or of the surrounding text. The complete indexing structure of a table consists of a forest of rooted category trees—two trees for a two Wang-category table, three trees for a three-Wang-category table, etc. Multicategory headers (like the two-category column header in Fig. 1) factor into a cross-product of header rows or columns. The height of the category trees depends on the minimum number of header columns or rows required to index the data cells.

The indexing structure can be exploited for searching relational DBMS and RDF triples. Although printed and HTML tables are logically symmetric in row and column organization, in relational tables indexing is asymmetric. Rows are *records* (or *tuples*), and columns are *fields* (or *attributes*). This distinction opens the way for a wealth of useful operations based on predicate logic and governed by the laws of relational algebra and calculus.

The fundamental property of a *header-indexed table* (*HIT*) is that every data cell is uniquely indexed by its row and column *header paths*, which are, respectively, left of and above the data region. A hierarchical (row or column) header may index one or more categories. A single-category header path consists of the root-to-leaf path of the corresponding category tree. A multicategory header path consists of concatenated category paths. Header-indexed tables are generally amenable to automated data extraction using only structural information.

Egregious tables (those that are not header-indexed) may not puzzle human readers [52], but they challenge algorithms and require external context to extract values with their applicable indexes. The genetic code tables in Fig. 3, for example, may have a much better layout for human understanding

RNA codon table

1st position	2nd position U	2nd position C	2nd position A	2nd position G	3rd position
U	Phe	Ser	Tyr	Cys	U
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	stop	stop	A
	Leu	Ser	stop	Trp	G
C	Leu	Pro	His	Arg	U
	Leu	Pro	His	Arg	C
	Leu	Pro	Gln	Arg	A
	Leu	Pro	Gln	Arg	G
A	Ile	Thr	Asn	Ser	U
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	U
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val	Ala	Glu	Gly	G
	Amino Acids				

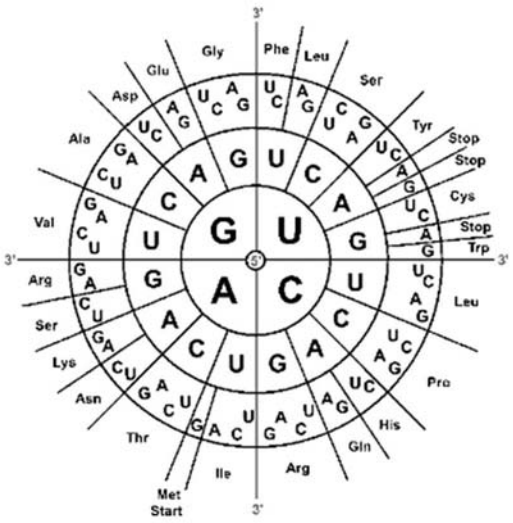

Fig. 3. Genetic coding tables. The table on the left is egregious because the second column of the row index is on the right. It can be converted to a HIT by moving the last column either to the left or the right of the first column. The table on the right (also a three-category table) presents the same data with radial indexing header paths.

than if they were laid out as HITs. Although it is easy for humans to recast such tables as HITs, the task is far from trivial for machines. The periodic table is a classic example: its layout succinctly captures element properties for an informed human reader. It can be cast into the layout of a HIT by listing the element symbols as row headers and providing column header labels for each of the depicted element properties. Egregious tables are relatively rare.

3.2 Header-indexed tables: formal characterization

Figure 4a shows a visual model of the HITswe process, which account for almost all human-readable tables (and even relational tables). The only essential spatial constraints are that the *RowHeader* must be to the left and aligned with the *Data* region, and that the *ColumnHeader* must be above and also aligned with the *Data* region. The remaining components are optional. The *TableTitle,* if included within the table, should be the topmost non-empty row. *Footnotes* along with their preceding *FootnoteMarkers* must be below the *RowHeader* and *Data* regions and cannot share their row with anything else. The corresponding reference to the footnote, matching the footnote marker, may occur in any cell above the footnote. *Notes,* which can occur anywhere, provide information about the source or dissemination of

the data (e.g., *Source: OECD* in Fig. 1). Duplicate rows and columns, including repeated row and column headers inserted to avoid scrolling, are detected and skipped. Empty rows or columns can be deleted without loss of information, yielding the simplified model in Fig. 4b.

Critical cells (*CC1*, *CC2*, *CC3*, *CC4*) delineate regions. As Fig. 4 shows, *CC1* and *CC2* demarcate the *StubHeader* and *CC3* and *CC4* demarcate the *Data* region. Furthermore, in combination with one another, these critical cells also demarcate both the *ColHeader* and *RowHeader* regions.

Letting row r_i and column c_i be the coordinates of critical cell *CCi*, a HIT satisfies the following constraints: $r_1 \le r_2 < r_3 \le r_4$ and $c_1 \le c_2 < c_3 \le c_4$. These constraints guarantee that the *ColHeader* and *RowHeader* regions properly align with the *Data* region and that the *Data* region is not degenerate. A single row ($r_3 = r_4$) or column ($c_3 = c_4$) of data is acceptable, provided both row and column headers exist. To complete our formalization of a HIT, we formulate region-level and cell-level constraints that provide a computable version of the visual representation of Fig. 4.

Region-level Constraints. The region-level spatial constraints can be formalized using a block algebra [67], which is a spatial application of Allen's interval algebra [68].

Figure 5 shows the 7 basic relations of interval algebra. The inverse relations interchange the roles of *x* and *y*: *xby* ≡ *y bi x, x m y* ≡ *y mi x*, etc. The row and column intervals of 2-D blocks are independent. Hence a constraint between any two blocks can be expressed as a pair of row and column constraints, as exemplified in Fig. 4b. If more than one horizontal or vertical relationship is possible, it is expressed as a disjunction, e.g., vertically, *TableTitle b* ∨ *m ColHeader*.

The constraints on a HIT are shown in a matrix form in Table 1. The relation pairs *(f i, b* ∨ *m)* appear in the row of *TableTitle* and column *ColHeader*. Further, the entry in the symmetric cell (row: *ColHeader*, column: *TableTitle*) will be its inverse, i.e., *f, bi* ∨ *mi*. Because of this symmetry, the cell entries in the gray region are not shown.

Cell-level Constraints. Apart from the region-level structural constraints, a HIT also satisfies the following cell-level constraints related to data cells, header cells, categories, and auxiliary cells comprising titles, notes, and footnotes.

Data cells

1. Each *DataCell* in a grid table is a singleton cell.
2. Every *DataCell* is indexed by header cells from every category.

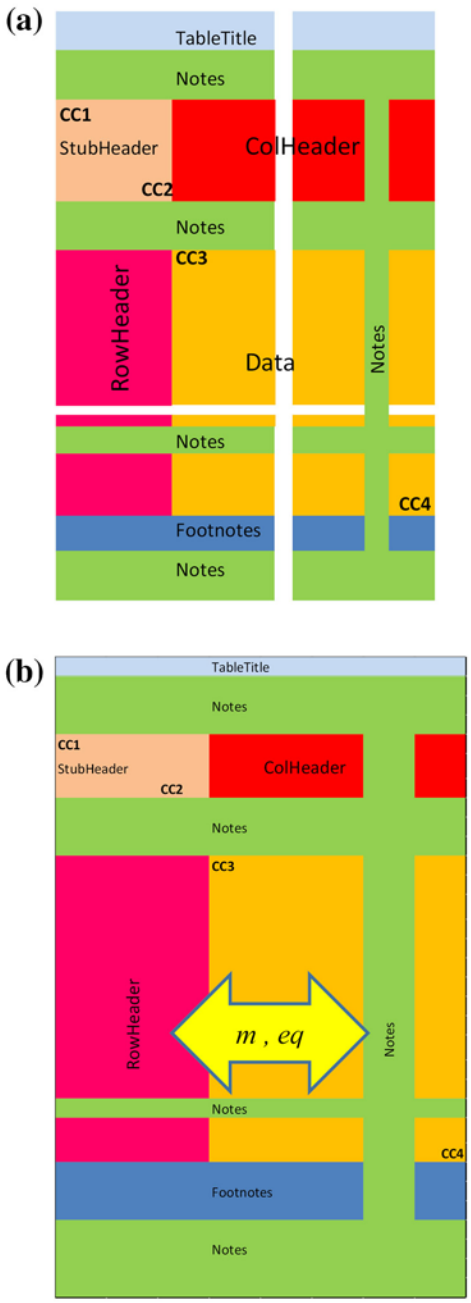

Fig. 4. Visual HIT model: **(a)** complete **(b)** simplified by removing all empty rows and columns to reduce size of constraint table. As an example of the *m, eq* constraint in the fourth row and last column of Table 1 below, RowHeader *meets* data horizontally, and is *equal* to data vertically.

Table 1. Spatial constraints of the header-indexed table model in Fig. 4b

	TableTitle	*StubHeader*	*ColHeader*	*RowHeader*	*Notes*	*Footnotes*	*Data*
TableTitle	eq, eq	si, b∨m	fi, b∨m	si, b	eq, m∨b	eq, b	fi, b
StubHeader		eq, eq	m, eq	eq, m∨b	s, m∨mi∨b	s, b	m, m∨b
ColHeader			eq, eq	mi, b∨m	f, m∨mi∨b	f, b	eq, m∨b
RowHeader				eq, eq	s, m∨b∨mi∨bi	s, b	m, eq
Notes					eq, eq∨bi∨b	eq, b∨bi∨mi	fi, bi∨mi∨b∨m
Footnotes						eq, eq	fi, mi
Data							eq, eq

The notation is based on Fig. 5. Each cell contains a horizontal constraint and a vertical constraint separated by a comma. Each constraint may have OR clauses indicated by ∨.

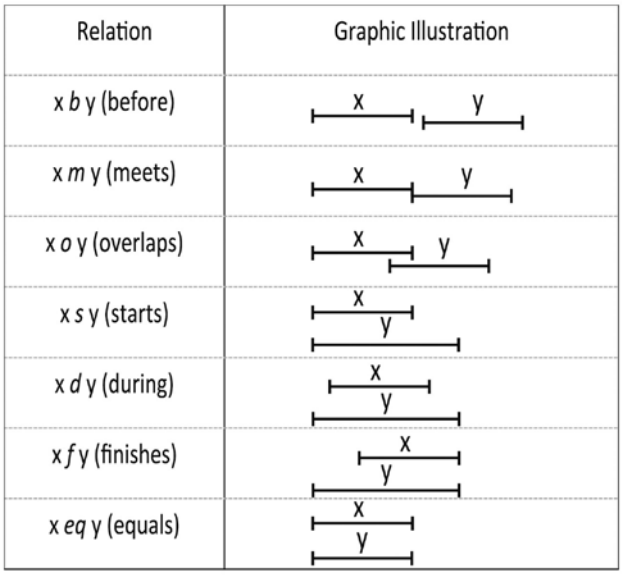

Fig. 5. The relations of Allen's interval algebra.

Header cells

1. Every *HeaderCell* belongs to at least one *HeaderPath*—a vertical sequence of cells through the column header or a horizontal sequence of cells through the row header.
2. *DataCell* (*r, c*) has *RowHeaderPath Cell*(*r,* c_1), …, *Cell*(*r,* c_2), where c_1 and c_2 are the column coordinates of *CC1* and *CC2*, i.e., the sequence of horizontal cells in the *RowHeader* region in row *r* ; and has *ColHeaderPath Cell*(r_1, *c*), …, *Cell*(r_2, *c*), where r_1 and r_2 are the row coordinates of *CC1* and *CC2*, i.e., the sequence of vertical cells in the *ColHeader* region in column *c*.
3. *Col*(*Row*)*HeaderPaths* (concatenations of *HeaderPaths* for multicategory headers) uniquely identify a column (row) of data cells.

Auxiliary cells

1. A footnote marker and its associated footnote may appear in a single cell or in two row-adjacent cells.
2. Every footnote marker has a footnote reference that may appear in the table title, header or data region.

In summary, the class of tables that we call HITs can be precisely specified in terms of computable spatial and logical constraints. We believe that HITs cover most printed, web, and spreadsheet tables, as well as relational database tables displayed in standard form with keys on the left. We shall now show that the above formalization makes HITs amenable to model-driven analysis.

4 Table region segmentation and cell classification

Segmentation consists of locating the critical "corner" cells *CC1* and *CC2* of the stub header, and *CC3* and *CC4* of the data region, as well as the rows or elementary cells containing the embedded table title, footnotes, footnote marks, footnote references, and miscellaneous notes. Our MIPS (Minimum Indexing Point Search) algorithm finds *CC1* and *CC2*. The underlying assumption is that the row headers (on the left) and column headers (above) index the data cells. Header indexing requires header cells to be aligned with the data cells they index, as is also required of HITs. Therefore MIPS transforms near-HITs into HITs by straightening out any "crooked" header paths by prefixing duplicate labels with unique labels.

Although *CC1* and *CC2* are found algorithmically, heuristics are needed to demarcate the top and bottom of the *data* region (indicated by *CC3* and *CC4*) from its surrounding regions. As shown in Sect. 4.3, the output of the segmentation and cell classification stage is a CSV *classification table* in a uniform format with one row for each cell of the source table.

4.1 Header segmentation

The input to the MIPS algorithm is a CSV table, converted from a web table. Figure 6 shows the first seven and last six rows of the exemplary table of Fig. 1 converted to CSV format and rendered as a table. Empty rows and columns are labeled as EMPTY (not shown in Fig. 6) to indicate that these rows and columns can be ignored during segmentation and classification. They are not deleted because that would interfere with referencing the original cell coordinates and because they sometimes serve as visual clues to focus on certain aspects of the table (e.g., Nordic countries in Fig. 1).

1 Official development assistance.										
Country	Million dollar					Percentage of GNI				
	2007	2008	2009	2010*	2011*	2007	2008	2009	2010*	2011*
Norway	3 735	4 006	4 081	4 580	4 936	0.95	0.89	1.06	1.1	1
Denmark	2 562	2 803	2 810	2 871	2 981	0.81	0.82	0.88	0.91	0.86
...										
...										
Australia	2 669	2 954	2 762	3 826	4 799	0.32	0.32	0.29	0.32	0.35
New Zealand	320	348	309	342	429	0.27	0.3	0.28	0.26	0.28
OECD/DAC1 c	104 206	121 954	119 778	128 465	133 526	0.27	0.3	0.31	0.32	0.31
1	DAC-countries are members of OECD's Development Assistance Committee.									
Source: OECD.										

Fig. 6. Part of the table of Fig. 1 in CSV grid table format that preserves the grid structure of the original HTML table. CCs *shaded yellow*.

We explain MIPS using the pseudo-code of Fig. 7, the table in Fig. 1, and the diagram of the search path for a slightly more complicated table in Fig. 8, As shown in the HIT model (Fig. 4b), the data region extends to the right of the table. MIPS operates on the portion of the table above the bottom of the data region whose rightmost bottom cell is indicated by *CC4*. This critical cell is found before MIPS is launched by searching from the bottom of the original table for the last row with a minority of empty cells (in Fig. 1, it is Row 30, with *OECD/DAC* in its first cell). Rows with at most a few empty cells are assumed to be part of the data region rather than notes or footnotes rows (which usually have only one or two non-empty cells).

Before the algorithm is called, empty cells resulting from splitting spanning cells are filled with the label of the spanning cell (like *MillionDollar* in Fig. 6). Duplicate labels (like "%"), if any, are prefixed with the preceding (to the left or above, respectively) unique labels (if available). Repetitive labels resulting from spanning cells are not considered duplicates. No prefixing is required for the exemplary table, but an example will be shown below.

The first while loop in Fig. 7 searches for the Minimum Indexing Point (MIP), which is the bottom right corner of cell CC2 = *(*R2,C2*)*. In Fig. 6 CC2 = *(*4, 1*)*. The algorithm finds the row header with the smallest number of columns that have no duplicate rows belowR2, and the column header with the smallest number of rows and no duplicate columns to the right of C2. The minimality property is local: (1) moving R2 up one cell or C2 left one cell would destroy the indexing property because the shorter column headers or narrower row headers will not be unique, and (2) moving R2 down or C2 to the right would destroy the minimality property because it adds unnecessary rows or columns. The global MIP (R2, C2) is *indexing,* locally *minimal*, and has the largest data area among the MIP candidates.

Figure 8 shows the search path followed by the MIPS algorithm of Fig. 7 on a hypothetical table. The search begins at the bottom left corner (at

MIPS Algorithm

 # MIPS locates the critical cells that demarcate the minimum row and column headers needed to index every data cell.

Input: CSV Table with ASCII cell strings, critical cell CC4
Output: critical cells CC1 and CC2 (*CC2 is the minimum indexing point*)

 # Initialize:
C_{max} ← last column of data cells; R_{max} ← last row of data cells *# from CC4*
R_1←1; C_1 ← 1; ….R_2←R_{max} – 1; C_2←1 *# (R_1, C_1) and (R_2, C_2) are the current CC1 and CC2 candidates*
Rightflag = Upflag = 0 *# these flags indicate whether R_1 or C_1 changed last.*
Max_area ← 0 *# for storing maximum data area so far*

 # Locate candidate MIPs by finding the minimum indexing headers:
while $C_2 < C_{max}$ and $R_2 \geq R_1$
 # "[... : ...]" denotes a rectangular region of the table.
if [R_2 + 1, C_1 : R_{max},, C_2] has no duplicate rows **and** [R_1,C_2 + 1 : R_2-1,C_{max}] has no duplicate columns,
 # i.e., candidate headers uniquely index the data rows to its right and the data columns below it
R_2 ← R_2 – 1 *# move CC2 up as long as indexing is preserved*
Upflag←1; Rightflag←0
else C_2 ← C_2 + 1 *# move CC2 right (moving right always preserves indexing)*
Rightflag ← 1
if Upflag = Rightflag = 1
Data_area ← (R_{max} – R_2 + 1) × (C_{max}–C_2 + 1) *# No. of data rows × columns*
If Data_area > Max_area
Max_area ← Data_area
CC2 ← (R_2, C_2) *# minimum indexing point with the largest data area so far*
Upflag←0

 # Locate CC1 at intersection of the top row and the leftmost column necessary for indexing
R_1 = 1, C_1 = 1
while [R_1 + 1, C_2 +1 : R_2, C_{max}] has unique columns, R_1←R_1+1
while [R_2 + 1, C_1 + 1 : C_2, R_{max}] has unique rows, C_1←C_1+1
return CC1 = (R_1,C_1), CC2 = (R_2,C_2)

Fig. 7. The MIPS algorithm\searches the input CSV table for minimum indexing points. During the first while loop the CC2 candidate moves up whenever it can, and to the right otherwise. Empty and duplicate rows and columns that extend over the whole table are tagged earlier and skipped. Header rows and columns with empty data, and data with empty header cells, are tagged as Notes. The provision for tagging trivial tables (only one data row or column) is not shown

Column 1 in the CC4 row) and moves up as long as both candidate header rows below and columns to the right are unique. When that condition is violated, the search turns to the right. The MIP must be located at an inside corner (right turn on the search path) where both the indexing and the minimality conditions are met.

There may be more than one inside corner along the search path. The (R2, C2) coordinates and area of the data region corresponding to a local

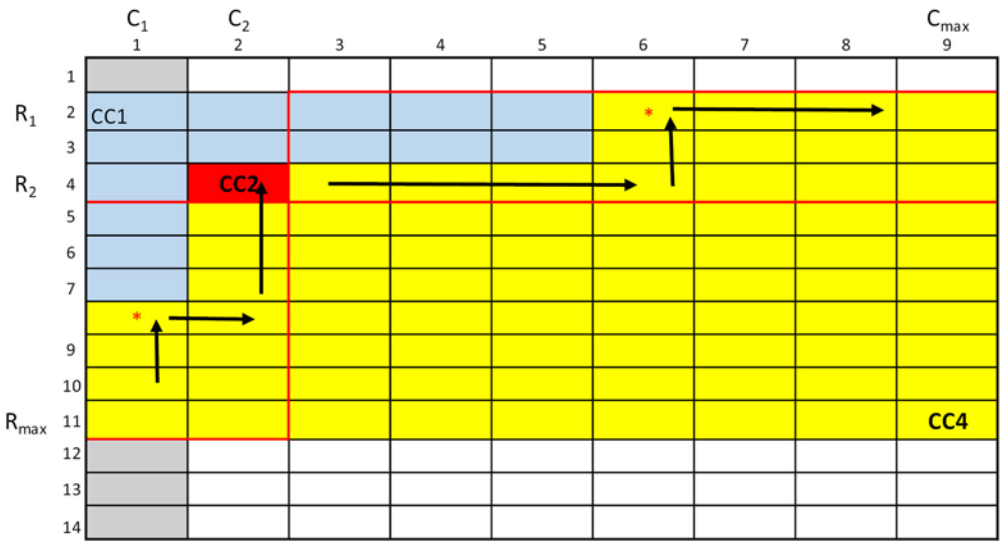

Fig. 8. An example with three local MIPs. The search path (*black arrows*) follows the boundary cells of the *yellow indexing region* to detect minimum indexing points at inside corners. The row and column headers are outlined in red. A *red asterisk marks* local MIPs. The global, MIP, i.e., cell (4, 2), is *shaded red*. Its data area is 49 cells, whereas the data areas of the other MIP are only 24 and 27. The critical cells are CC1 = *(2, 1)* and CC2 = *(4, 2)*. Therefore the stub header is [R1,C1, : R2,C2]. = *(2, 1 : 4, 2)*. The first row will be designated as table title in a subsequent step, and the bottom rows will become notes or footnotes. This figure does not show empty rows and columns beyond the actual table, which are detected and bypassed.

MIP is recorded if the area exceeds the current maximum. After the algorithm completes the search from the bottom left corner to the top right corner, the MIP with the largest data area becomes CC2 (searching from the top right would work equally well).

CC2 determines only the rightmost column of the row header and the bottom row of the column header. In the last two while loops, CC1 is found by deleting the rows above the column header and the columns left of the row header that are not necessary for indexing the data region.

In the table of Fig. 1 all the headers are properly aligned, so all that is required is distributing the labels into the atomic cells resulting from fragmented spanning cells. But Fig. 9 shows an example where it is necessary to prefix the labels of some header cells. This table is not a HIT because it violates the header-cell-uniqueness constraint of a HIT. Prefixing converts it into a HIT by inserting a row with unique predecessor labels before the duplicate labels.

Over 15% of the tables in our collection require prefixing to turn them into HITs. Unlike the example in Fig. 9, most of them are in row headers.

Table 9. Numbers of outgoing short messages and multimedia messages from mobile phones in 2002-2008					
Year	Short messages, thousands 1)	Change, %	Short messages/ subscription	Multimedia messages, thousands	Change, %
2003	1 647 218	24,3	347	2 314	
2004	2 193 498	33,2	439	7 386	219,2

Fig. 9. Part of a web table that requires prefixing. The duplicate labels "Change %" become unique after being prefixed as: Short messages, thousands 1)/Change % and Multimedia messages, thousands/Change %.

After this prefixing step and the analogous step on the transposed rows, the MIPS algorithm proceeds as explained.

MIPS finds only CC1 and CC2. Then the program checks the original table under the column header candidate to find CC3 as the leftmost cell of the first filled row of data region. CC4, was already located earlier as the right-most cell of the last filled row. The cells in the corresponding regions are then labeled StubHeader, RowHeader, ColHeader, or Data.

4.2 Auxiliary regions

Table titles are almost invariably in a spanning cell at the top of a table, therefore all the cells of the topmost non-empty row are labeled *TableTitle*. Footnote markers, if present, are found by searching below the data region for a list of common footnote-mark symbols (*, #, . ◦, †, etc.) and for single digits and letters (possibly followed by a period or a parenthesis). They are labeled *FNprefix*. All the cells following a footnote marker in the same row are marked *FNtext*. A cell containing both a *FNprefix* and a *FNtext* is marked *FNprefix&FNtext*. The program searches the entire table above the footnotes for the already detected and isolated footnote markers. If the footnote reference is found, the cell is labeled *FNref* (if the footnote reference is in a cell by itself) or *X&FNref*, where *X* can be any of the table regions above the footnote region, e.g., *RowHeader& FNref* for the last cell of the row header in Fig. 1. Here our program missed the footnote reference "1" because it is embedded in the middle of the header label OECD/ DAC1 countries total, and of course its superscript formatting disappeared in CSV.

Finally, every cell in a row that contains only non-empty cells that have not been otherwise classified is labeled Note.

4.3 Cell classification

The output of this stage is a *Classification Table*, e.g., Fig. 10 for the table in Fig. 1. This table is in a five-column format, with a row entry (after the header row) for each cell of its source table. The first column is a unique

Cell_ID	Row	Column	Content	Class
ODA_R1_C1	1	1	1 Official development assistance.	tabletitle
ODA_R1_C2	1	2		tabletitle
ODA_R1_C3	1	3		tabletitle
ODA_R1_C4	1	4		tabletitle
ODA_R1_C5	1	5		tabletitle
ODA_R1_C6	1	6		tabletitle
ODA_R1_C7	1	7		tabletitle
ODA_R1_C8	1	8		tabletitle
ODA_R1_C9	1	9		tabletitle
ODA_R1_C10	1	10		tabletitle
ODA_R1_C11	1	11		tabletitle
ODA_R2_C1	2	1		EMPTY
ODA_R2_C2	2	2		EMPTY
ODA_R2_C3	2	3		EMPTY
ODA_R2_C4	2	4		EMPTY
ODA_R2_C5	2	5		EMPTY
ODA_R2_C6	2	6		EMPTY
ODA_R2_C7	2	7		EMPTY
ODA_R2_C8	2	8		EMPTY
ODA_R2_C9	2	9		EMPTY
ODA_R2_C10	2	10		EMPTY
ODA_R2_C11	2	11		EMPTY
ODA_R3_C1	3	1	Country	stubheade
ODA_R3_C2	3	2	Million dollar	colheader
ODA_R3_C3	3	3		colheader
ODA_R3_C4	3	4		colheader
ODA_R3_C5	3	5		colheader
ODA_R3_C6	3	6		colheader
ODA_R3_C7	3	7	Percentage of GNI	colheader

Fig. 10. First 30 rows of the 408-row classification table for the table of Fig. 1.

cell identifier with the file name of the CSV table and the cell coordinates. The second and third row give the numerical cell coordinates separately for ease of handling. The fourth column is the content of the cell in the original table, and the last column is its assigned class. Section 7 contains some examples of the application of this table.

5 Complex header structures

Among our 400 tables, over 30% have complex header structures—multiple row column headers, multiple-column row headers, and single row (column) headers that require prefixing. We analyze all the headers to discover their category structure, and we use the discovered structure to create canonical relational tables which are searchable with standard database query languages.

5.1 Category analysis

We define a simple algebra over the set of header labels. Each label appearing in a header is said to cover a subset of the cells in a table's data region.

For example, in Fig. 1 the label *Million dollar* covers the first five columns of data cells and the label 2007 covers the first and the sixth columns. We define two binary operations, × (intersection) and + (union) over the header labels with respect to their covering properties. For example, the expression Million dollar+Percentage of GNI covers all the columns of the data region, while Million dollar ×2007 covers only the first column. In this formulation, each header path can be equated with the product of labels appearing in it, and the set of all header paths can be equated with a sum of products (SOP) expression, in which each product term corresponds to a unique header path.

To determine the number of categories and their hierarchical structures, a factorization of an SOP expression *E* is carried out under the following constraints:

1. Only the distributive law and the associative laws are used. The × operation has higher precedence than +.
2. The commutative law is disallowed, so that ordering is maintained both among header paths for + and within header paths for ×. To avoid changing the number and length of paths, the idempotency laws are also disallowed.
3. The factorization preserves the unique indexing property of E.

The factorization is complete in the sense that none of its terms can be factored further.

5.2 Factorization algorithm

Figure 11a shows the column header of a table in our collection. In Fig. 11b, the lengthy cell labels are replaced by alphabetic symbols to shorten the algebra. Figure 12 presents a formal description of the recursive algorithm for the factorization of header paths. *E* is a *sum of products* (SOP) algebraic expression where × denotes vertical concatenation and + denotes horizontal concatenation of table cells. For the column header shown in Fig. 11b,

$$E = a \times c \times d + a \times c \times e + a \times c \times f + b \times c \times d + b \times c \times e + b \times c \times f$$

The output of *Fact(E)* is the header factored into one or more Wang categories. In the first pass of the factorization, the product terms of *E* are scanned from left to right, factoring out common *prefix* (first) symbols, producing corresponding *suffix* SOP expressions:

$$E = a \times (c \times d + c \times e + c \times f) + b \times (c \times d + c \times e + c \times f)$$

2006	2006	2006	2007	2007	2007
Government transfers	Government transfers	Government transfers	Government transfers	Government transfers	Government transfers
Average $ constant 2007	Implicit transfer rates1 %	Shares %	Average $ constant 2007	Implicit transfer rates1 %	Shares %

(a)

a	*a*	*a*	*b*	*b*	*b*
c	*c*	*c*	*c*	*c*	*c*
d	*e*	*f*	*d*	*e*	*f*

(b)

Fig. 11. Example column header to illustrate recursive factorization. **(a)** Column header of table T120 in our collection; **(b)** Equivalent representation with the cell labels replaced by *letter symbols*.

In the second pass, the resulting expression is scanned again from left to right, to factor out common *suffixes*, producing simple sums of prefixes that multiply them:

$$E = (a + b) \times F, \text{ where, } F = (c \times d + c \times e + c \times f)$$

In general, after the two passes, E is decomposed into the following form:

$$E = S_1 \times F_1 + S_2 \times F_2 + \cdots + S_n \times F_n$$

where each S_i is a simple sum of prefixes (degenerately, a singleton) and each F_i is an SOP simpler than E. After the second pass, *Fact(E)* recursively calls itself with F_i's as the arguments and returns the factorization as:

$$E = S_1 \times Fact(F_1) + S_2 \times Fact(F_2) + \cdots + S_n \times Fact(F_n)$$

For the example header, the recursive call *Fact(F)* results in the factorization:

$$F = c \times (d + e + f)$$

with resulting factorization of the original expression:

$$\begin{aligned} E &= (a + b) \times c \times (d + e + f) \\ &= (2006 + 2007) \times \text{Government transfers} \times (\text{Average \$ constant } 2007 + \text{Implicit transfer rates1\% + Shares\%}) \end{aligned}$$

showing the two non-degenerate categories {*a, b*} and {*d, e, f* } and the degenerate category {*c*}.

```
Fact(E):
    X = ""      # Initialize to null string
    if E is a simple sum:                                   # simple sum: a literal or sum of literals
        X = E
    else:                                                   # sum of products
        factor out common prefix and factor out common suffix   # Passes 1 & 2
            to find the decomposition, E = S1×F1 + S2×F2 + ... + Sn×F n
        for i = 1 to n-1:                                   # Loop not executed if n = 1
            X = Cat(X , Si, "x", Fact(Fi), "+")             # Cat is string concatenation
        X = Cat(X, Sn , "×", Fact(Fn))                      # Accumulate last term of the decomposition
    return X
```

Fig. 12. The factorization algorithm to determine the category structure of table headers.

5.3 Category tables

The table designer's choice of rows or columns for laying out the categories depends primarily on the number of leaf nodes in the category tree and on the size and aspect ratio of the available space. In relational tables, however, rows are tuples (records in Access), while columns are attributes (fields in Access). The database schema immutably assigns the values of each category to either a record or a field. We introduce category tables to represent the data elements in "ordinary" tables within the constraints of relational tables.

Our category table is a relational table where each row comprises the indexing header paths and the corresponding indexed data value. Therefore the number of rows in the category table equals the number of data cells in the original table (plus one for the relational table's field names in a header row). The number of columns is one for the *Cell_ID*, plus one for *DATA*, plus the sum of the heights of the category trees (which, usually, equals the sum of the column width of the row header and row height of the column header). For our exemplary table, the category table has 240 rows and 5 columns.

In the category table, Cell_ID is a key field and each cell label in the original header paths becomes a key field value in the composite key comprising all the category fields. The data values become non-key field values. Figure 13 shows part of the category table for the exemplary table. The first column references the original (table, row, and column) location of each data cell. The row headers in Fig. 1 are values in the RowCat_1.1 column in Fig. 13, and the column headers are distributed as values in the ColCat_1.1 and ColCat_2.1 columns according to their factorization—values in ColCat_1.1 from the factor (*Million dollar* + *Percentage of GNI*) and values in ColCat_2.1 from the factor (*2007* + *2008* + *2009* + *2010** + *2011**).

Cell_ID	RowCat_1.1	ColCat_1.1	ColCat_2.1	DATA
ODA_R6_C2	Norway	Million dollar	2007	3735
ODA_R6_C3	Norway	Million dollar	2008	4006
ODA_R6_C4	Norway	Million dollar	2009	4081
ODA_R6_C5	Norway	Million dollar	2010*	4580
ODA_R6_C6	Norway	Million dollar	2011*	4936
ODA_R6_C7	Norway	Percentage of GNI	2007	0.95
ODA_R6_C8	Norway	Percentage of GNI	2008	0.89
ODA_R6_C9	Norway	Percentage of GNI	2009	1.06
ODA_R6_C10	Norway	Percentage of GNI	2010*	1.1
ODA_R6_C11	Norway	Percentage of GNI	2011*	1
ODA_R7_C2	Denmark	Million dollar	2007	2562
ODA_R7_C3	Denmark	Million dollar	2008	2803
ODA_R7_C4	Denmark	Million dollar	2009	2810
ODA_R7_C5	Denmark	Million dollar	2010*	2871
ODA_R7_C6	Denmark	Million dollar	2011*	2981
ODA_R7_C7	Denmark	Percentage of GNI	2007	0.81
ODA_R7_C8	Denmark	Percentage of GNI	2008	0.82
ODA_R7_C9	Denmark	Percentage of GNI	2009	0.88
ODA_R7_C10	Denmark	Percentage of GNI	2010*	0.91
ODA_R7_C11	Denmark	Percentage of GNI	2011*	0.86
ODA_R8_C2	Finland	Million dollar	2007	981
ODA_R8_C3	Finland	Million dollar	2008	1166
ODA_R8_C4	Finland	Million dollar	2009	1290
ODA_R8_C5	Finland	Million dollar	2010*	1333
ODA_R8_C6	Finland	Million dollar	2011*	1409
ODA_R8_C7	Finland	Percentage of GNI	2007	0.39
ODA_R8_C8	Finland	Percentage of GNI	2008	0.44
ODA_R8_C9	Finland	Percentage of GNI	2009	0.54
ODA_R8_C10	Finland	Percentage of GNI	2010*	0.55
ODA_R8_C11	Finland	Percentage of GNI	2011*	0.52

Fig. 13. Category table for the table in Fig. 1 (first 30 of 240 rows).

The combined row and column headers that uniquely index each data value in the DATA column also index the data values in the original table. Because "ordinary" tables can always be recast as category tables, the formulation of the category table format and the automated transformation of HITs to category tables make a significant contribution to importing tabular web content into structured and searchable relational data structures. Moreover, as we show in Sect. 7, category tables also provide a direct path to the formulation of RDF triples and thus to searchable semantic-web content.

6 Experimental results

200 HTML tables (*Troy 200*) were randomly drawn from a set of tables collected earlier from large statistical Web sites in the USA and abroad [69]. The geopolitical and research sources included Statistics Canada, Science Direct, The World Bank, Statistics Norway, Statistics Finland, US Department of Justice, Geohive, US Energy Information Administration, and US Census Bureau.

Table 2. Experimental results

Observations	*Corpus*	
	Troy 200	*SAUS 200*
Number of tables	200	200
Successfully processed	199	198
Only one row or col of data	1	2
Errors		
Minimum indexing point (MIP)	2	2
Critical cells (CCs)	4	9
Gross size of tables		
Rows average	25	64
Maximum	183	453
Columns average	11	17
maximum	80	81
Cells average	290	1184
Maximum	7320	15,094
Net size of tables		
Data rows (average)	15	45
Data columns (average)	5	15
Data cells (average)	85	676
Categories		
Multicategory row headers	7	12
Multicategory column headers	14	13
Prefixed headers		
Row headers	23	63
Column headers	3	0
Size of headers		
1-col row header and 1-row col header	145	56
Row headers w. 3 or more columns	1	9
Column headers w. 3 or more rows	3	44
Footnotes		
Footnoted tables	56	NA
Reference markers (total)	91	NA
References found (total)	158	NA
References not found (total)	15	NA
Notes		
Rows (average)	5.13	8
Columns (average)	0.06	0.89
Run time (seconds) w/o file output	15.6	61.9

We also tested our program on 200 spreadsheet tables (*SAUS 200*) randomly selected from a published data set of over 1300 spreadsheet tables from the Statistical Abstract of the United States (SAUS) posted by Michel Cafarella [49,70]. For each workbook, we only converted the first data sheet that contained a table without the footnotes. Table 2 shows the results reported by our program on all 400 tables. The SAUS tables are larger than

Table 3. Joint distribution of minimum indexing row and column header sizes in the original (non-prefixed) tables

|RH|	|CH|			
	1	2	3	4
1	56	35	8	0
2	43	12	0	0
3	24	7	0	0
4	4	1	1	0
5	7	0	0	0

the Troy tables, with about twice as many rows and columns and four times as many cells. Many tables have over 100 rows or columns.

The critical cells obtained by our program were verified against the ground truth obtained with *VeriClick* [61]: for each table, the four critical cells that demarcate the minimum indexing headers and the data region were identified.

One of the 200 Troy tables was found to be trivial, having only one data column. Of the 199 non-trivial HTML tables, the MIP (CC2) was correctly located in 197 tables. All four CC errors were caused by notes-data confusions, such as rows or columns filled with blanks or periods or X's that did not exhibit enough variety to qualify as data, or to rows with a variety of units that were mistaken for data.

The corresponding numbers for the SAUS spreadsheets were 2 unprocessed trivial tables, 2 MIP errors, and 9 tables (including the above two) with errors in some critical cells. Seven of the nine miss-segmentations were caused by notes-data confusions. One header had an unprefixable duplicate label by mistake (a source error). Indexing of another column header failed because the appropriate prefix was to the *right* of a duplicate label. The overall segmentation accuracy, excluding trivial tables, was *(*195 + 189*)/*397 = 96.7%.

Table 3 shows the distributions of the 198 non-trivial SAUS row and column header sizes. The data shows that multirow column headers are more frequent (99) than multicolumn row headers (64). The statistics on header sizes, prefixed rows and columns, number of row and column categories, and number of notes rows are based on analysis of the minimal indexing headers found by MIPS that do not depend on subjective interpretation of the table.

Different ground truth could be formulated to include rows redundant for indexing above this minimal column header. One could also justify including in the column header some redundant rows (for example, units) above the data region. Options for expanding headers are under investigation.

All 46 multicategory row and column headers were determined correctly by factoring after prefixing when required. Only one table had both multiple row and column categories. Prefixing is more prevalent in row headers where hierarchies are usually indicated by indentation or distinctive type style rather than additional columns. Of the 397 processed tables, 89 required row or column prefixing. Only one table required two levels of row prefixing.

The footnotes were checked only on the Troy tables because in SAUS the footnotes were on separate worksheets. All the footnotes were found in the 56 tables that had them, but not all the references to them. The program detected 158 reference marks to the footnotes within the body of the tables (some had more than a dozen). It missed 15 in three tables. Superscripts are not retained in CSV files.

Processing the Troy tables, excluding writing the 199 files for category tables and the 199 classification files, required only 16 seconds on a 2.4 GHz Dell Optiplex 7010 with 8GB RAM running Python 2.7 under Windows 7.0. The larger SAUS tables took 62 s on the same platform.

7 Application queries

Having shown how to transform a human-readable table to a machine-readable table, we now demonstrate that the transformations yield directly useable information for formal queries in widely available application software. Such a "proof of the pudding" is seldom offered in prior work where the table-processing results are usually retained only in an ad hoc format.

We process queries using industry standards—Microsoft Access for SQL queries over a generated relational database and the OpenLink Virtuoso semantic-web endpoint and Protégé for SPARQL queries over a generated triple store represented in the semantic-web languages RDF [71] and OWL [72]. In all cases the generated, canonical category tables and the generated classification tables are automatically imported into an appropriate data store where their content can be queried directly. Before importing them, an automated editing pass over cell content replaces decimal commas with periods and deletes thousands-separator blanks and commas (as in Fig. 13). To accommodate syntax requirements, the dots in *RowCat* and *ColCat* identifiers are also removed.

The query in Fig. 14 computes the GNI for every country for every year from the category table in Fig. 13. Figure 15 shows partial results.

A second query illustrates combining disparate, but semantically overlapping tables. The table in Fig. 16 quantifies international trade by land through Detroit, Michigan, and another table in our test set quantifies and compares US trade with its NAFTA partners, Canada and Mexico. Its "U.S.

```
SELECT MDollarTbl.RowCat_11 AS Country, MDollarTbl.ColCat_21 AS Year,
FORMAT(MDollarTbl.DATA, '#,###') AS MDollarAmt,
FORMAT(PrcntTbl.DATA, '#.##') AS PrcntGNI,
FORMAT(100000000*MDollarTbl.DATA/PrcntTbl.DATA, '###,###,###,###') AS GNI
FROM ODA_Mx1 AS MDollarTbl, ODA_Mx1 AS PrcntTbl
WHERE MDollarTbl.RowCat_11 = PrcntTbl.RowCat_11
   AND MDollarTbl.RowCat_11 <> 'EMPTY' AND MDollarTbl.RowCat_11 NOT LIKE 'OECD*'
   AND MDollarTbl.ColCat_11 = 'Million dollar' AND PrcntTbl.ColCat_11 LIKE 'Percentage*'
   AND MDollarTbl.ColCat_21 = PrcntTbl.ColCat_21;
```

Fig. 14. Access SQL query to compute GNI for every country in the ODA able of Fig. 1.

Query1

Country	Year	MDollarAmt	PrcntGNI	GNI
Norway	2007	3,735	.95	393,157,894,737
Norway	2008	4,006	.89	450,112,359,551
Norway	2009	4,081	1.06	385,000,000,000
Norway	2010*	4,580	1.1	416,363,636,364
Norway	2011*	4,936	1.	493,600,000,000
Denmark	2007	2,562	.81	316,296,296,296
Denmark	2008	2,803	.82	341,829,268,293

Fig. 15. MS-Access screenshot of results of Query 1 (partial).

	A	B	C	D	E	F	G	H
1	TABLE 4. Value of International Land Trade via Detroit, MI, by Mode: 1999-2003							
2								
3	($ millions)							
4								
5	Excel | CSV							
6								
7		1999	2000	2001	2002	2003		
8	Truck	83,889	85,468	79,762	85,062	84,811		
9	Rail	8,343	8,598	11,909	15,607	16,723		
10	Pipeline	45	78	67	50	92		
11	Other and	306	297	244	172	263		
12	Total	92,583	94,441	91,982	100,891	101,890		

Fig. 16. Table C10028 in our test set: International land trade with the USA through Detroit, Michigan.

surface trade" column over several years enables a query across the tables that finds the percent of US land trade through Detroit vs. the surface trade with NAFTA partners for all the years the two tables have in common. For the year 1999, for example, the Detroit land trade was 18.5% of the land trade with NAFTA partners.

```
<rdf:RDF
   xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
   xmlns:T028Mx1='http://osm.cs.byu.edu/tables/T028Mx1-rdf#'
>
<rdf:Description rdf:about='C10028_R8_C2'>
   <T028Mx1:RowCat_11>Truck</T028Mx1:RowCat_11>
   <T028Mx1:ColCat_11>1999</T028Mx1:ColCat_11>
   <T028Mx1:DATA>83889</T028Mx1:DATA>
</rdf:Description>
<rdf:Description rdf:about='C10028_R8_C3'>
   <T028Mx1:RowCat_11>Truck</T028Mx1:RowCat_11>
   <T028Mx1:ColCat_11>2000</T028Mx1:ColCat_11>
   <T028Mx1:DATA>85468</T028Mx1:DATA>
</rdf:Description>
...
</rdf:RDF>
```

Fig. 17. Generated RDF triples. The first triple is (*C10028_R8_C2, RowCat_11, Truck*), the second is (*C10028_R8_C2, ColCat_11, 1999*), and the third is (*C10028_R8_C2, DATA, 83889*), which altogether means that the cell identified by *C10028_R8_C2* (the cell in Table C10028 displayed in Fig. 16 at Row 8 and Column 2) has row header *Truck*, column header *1999*, and data value *83889*.

Queries over category tables require that query writers know the row and column categories of the tables. A third SQL query applies to classification tables (e.g., Fig. 10), which are independent of category structure. Classification tables contain the meta-information needed for further downstream processing in automating table interpretation such as identifying aggregate operations. The third query checks for one of the most common aggregate-operation configurations: a row of data values labeled *Total* whose corresponding column data values sum to the total values. Interestingly, the query found several discrepancies with actual totals not matching stated totals, e.g., the 2003 column in Fig. 16.

To produce semantic-web data for queries, we create RDF triples—(subject, predicate, object) statements (Fig. 17). As an illustration of querying semantic-web data, Fig. 18 gives a SPARQL query for the land-trade query above.

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX T028Mx1:<http://T028Mx1-rdf#>
PREFIX T079Mx1:<http://T079Mx1-rdf#>

SELECT xsd:int(?Year) AS ?Year,
   xsd:int(?Amount) AS ?MillionDollarAmt,
   xsd:int(?Value) AS ?BillionDollarAmt,
   100*xsd:float(?Amount)/(xsd:float(?Value)*1000) AS ?PercentDetroitLandTrade
FROM <http://osm.cs.byu.edu/tables/T028.rdf>
FROM <http://osm.cs.byu.edu/tables/T079.rdf>
WHERE {
?DetroitLandTradeCell  T028Mx1:RowCat_11  ?TransportMode .
?DetroitLandTradeCell  T028Mx1:ColCat_11  ?Year .
?DetroitLandTradeCell  T028Mx1:DATA  ?Amount .
?USGoodsTradeCell  T079Mx1:RowCat_11  ?Year2 .
?USGoodsTradeCell  T079Mx1:ColCat_11  ?TradeType .
?USGoodsTradeCell  T079Mx1:DATA  ?Value .
FILTER (
   regex(?TradeType, "surface trade.*current.*dollars")
   AND regex(?TransportMode, "Total")
   AND ?Year = ?Year2
   )
}
```

Fig. 18. SPARQL query.

The SPARQL query formulated above requires some knowledge of the queried table. In Fig. 18, for example, we see the line *?DetroitLandTradeCell T028Mx1:RowCat_11 ?TransportationMode*. Formulating this line (and some others) of the query requires understanding the structure of input tables. To remove structure dependencies for global queries, we programmed the construction of a uniform set of triples based on the canonical category tables. While in the triple construction described above the number of triples for each cell depends on the category structure of each table, the uniform OWL model triples do not. Instead, each cell is described by the same number of triples (based on the widest of the category tables). Hence, all of our tables can be searched simultaneously with a single query, for example, to determine in which tables *Exports* appears as a column category. Because of the uniformity of the model, the query (with prefix headers omitted) simplifies to:

```
Select distinct ?cell ?value where{
?cell table:hasColumn ?col filter regex(?col, Exports).
?cell table:hasValue ?value}
```

In Protégé on a Lenovo T61 laptop, this query executed in a fraction of a second over a 104 megabyte triple store.

8 Conclusion

The formalization of header-indexed tables (HITs) by means of block algebra and cell constraints models the table layouts that cover the vast majority of tables encountered in print and on the web. It obviates previous attempts to recognize their infinite variety of framing, partial ruling, typeface, color scheme, or cell formatting details. The formalization serves as the basis for indexing and factoring algorithms that convert human-readable HITs into a machine-processable form. Importing the transformed web tables into either a relational database or an RDF/OWL triple store enables them to be queried with SQL or SPARQL. Moreover, the HIT formalization encompasses auxiliary information: table titles, footnote components, and miscellaneous notes, broadening previously reported work.

The HIT formalization not only engenders an algorithmic solution to discovering indexing headers and finding their multicategorical indexing structure, but also provides a target for processing tables that do not strictly satisfy the HIT definition. As shown in Sect. 4, prefixing converts tables with "crooked" header indexes into bona fide HITs.

The proposed algorithms are based on a formal definition of header-indexed tables. Thus they need no statistically significant experimental

validation, only a demonstration of implementability and applicability. Although tables on the web are not always well formed, most are or can be converted (e.g., through prefixing) to be so. In our small but heterogeneous collection of 200 web tables, MIPS found all but two of the minimum indexing points and correctly segmented 98% of the minimal table headers and the data regions. Fact discovered all 21 multicategory headers. The heuristics for table titles, notes, and footnotes probe the limits of purely syntactic table processing. The category and classification tables were imported and queried in Access, Virtuoso, and Protégé. The tables and the critical-cell ground truth, already in use by other researchers, will be posted at the IAPR TC-11Web site.

The breadth of our definition of header-indexed tables was confirmed by running our program on 200 spreadsheet tables posted by others. All 25 multicategory headers were found. Many of the spreadsheets have truly puzzling headers and layouts, yet our program correctly segmented all but nine. Only one error was caused by a table that violates the HIT postulates (by a repeated header); the other 8 errors were data/notes/units confusions. For further improvement, we could either make our program more robust to unexpected features like columns containing only detached footnote references, rows, or columns of identical data or unusual symbols, and misplaced headers, or turn to more source-specific information like formatting conventions and domain semantics. Given how few errors are left, evaluating either option will require ground-truthing much larger and more varied collections of tables, or developing downstream applications that provide useful feedback.

This research also sets the stage for other near-future work. In addition to enabling formal queries, the cell classification table tags each cell of every processed table according to its function in the table. Knowing the cell classification and the category-tree indexing structure is likely to aid discovering the scope of aggregate operations and the operands of simple arithmetic operations, typing data values, and discovering implicit roots of category trees. Without meaningful category labels for every category, we cannot really claim that we *understand* tables. Resolving these issues will require matching table facets and features with semantic resources, whereas our work here is based on syntactic analysis. Longer-term research objectives include (1) interpreting tables with fully resolved syntax and semantics, (2) turning egregious tables into HITs, (3) integrating interpreted tables into ontologies, and (4) automating free-form query processing over collections of interpreted and integrated table content. All of this will require continuing efforts to combine the perspectives of the document-processing, information retrieval, database, and web-science communities.

Acknowledgments — Mukkai Krishnamoorthy acknowledges the help of Dr. Ravi Palla with Protégé. Prof. Andreas Dengel (DFKI) gave us excellent advice not only for improving the presentation but also for one of the algorithms.

References

1. Cafarella, W.J., Halevy, A., Wang, D.Z., Wu, E. , Zhang, Y.: Webtables: exploring the power of tables on the web. In: VLDB '08, Auckland, New Zealand (2008)
2. Galkin, M., Mouromtsev, D., Auer, S.: Identifying web tables—Supporting a neglected type of content on the web. In: International Conference on Knowledge Engineering and Semantic Web (KESW). arXiv:1503.06598 [cs.IR] (2015)
3. Wang, X.: Tabular abstraction, editing, and formatting, Ph.D. thesis, University of Waterloo (1996)
4. Frier, B.: Roman life expectancy: Ulpian's evidence. Harv. Stud. Classic. Philol. **86**, 213–251 (1982)
5. Zanibbi, R., Blostein, D., Cordy, J.R.: A survey of table recognition. Int. J. Doc. Anal. Recognit. **7**(1), 1–16 (2004)
6. Laurentini, A., Viada, P.: Identifying and understanding tabular material in compound documents. In: Proceedings of the Eleventh International Conference on Pattern Recognition (ICPR'92), The Hague, pp. 405–409 (1992)
7. Turolla, E., Belaid, Y., Belaid, A.: Form item extraction based on line searching. In: Kasturi, R., Tombre, K. (eds.) Graphics Recognition—Methods and Applications. Lecture Notes in Computer Science, vol. 1072, pp. 69–79. Springer, Berlin (1996)
8. Chandran, S., Kasturi, R.: Structural recognition of tabulated data. In: Proceedings of the Second International Conference on Document Analysis and Recognition (ICDAR'93), Tsukuba Science City, Japan, pp. 516–519 (1993)
9. Itonori, K.: A table structure recognition based on textblock arrangement and ruled line position. In: Proceedings of the Second International Conference on Document Analysis and Recognition (ICDAR'93), Tsukuba Science City, Japan, pp. 765–768 (1993)
10. Pinto, D., McCallum, A., Wei, X., Croft, W.B.: Table extraction using conditional random fields. In: Proceedings of the 26th Annual International ACM Y. SIGIR Conference on Research and Development in Information Retrieval, pp. 235–242 (2003)
11. Hirayama, Y.: A method for table structure analysis using DP matching. In: Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR'95), Montreal, Canada, pp. 583–586 (1995)
12. Handley, J.C.: Document recognition. In: Dougherty, E.R. (ed.) Electronic Imaging Technology, chap. 8. SPIE—The International Society for Optical Engineering (1999)

13. Zuyev, K.: Table image segmentation. In: Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'97), pp. 705–708 (1997)
14. Cesarini, F., Marinai, S., Sarti, L., Soda, G.: Trainable table location in document images. Procs. 16th Int'l Conf on Pattern Recognition **3**(236–240), 2002 (2002)
15. Wang, Y., Hu, J.: A machine learning approach to table detection on the web. In: WWW Conference, Honolulu, pp. 242–250 (2002)
16. Abu-Tarif, A.: Table processing and table understanding, Master's thesis, Rensselaer Polytechnic Institute, May (1998)
17. Rastan, R., Paik, H.-Y., Shepherd, J.: TEXUS: A task-based approach for table extraction and understanding. In: Proceedings of the ACM Conference on Document Engineering, Lausanne, vol. 15, pp. 25–34, Sept (2015)
18. Pyreddy, P., Croft, W.B.: TINTIN, a system for retrieval in text tables. Technical Report UM-CS-1997-002, University of Massachusetts, Amherst (1997)
19. Kieninger, T.G.: Table structure recognition based on robust block segmentation. In: Proceedings of Document Recognition V (IS&T/SPIE Electronic Imaging'98), San Jose, CA, vol. 3305, pp. 22–32 (1998)
20. Hu, J., Kashi, R., Lopresti, D., Wilfong, G.: Table structure recognition and its evaluation. In: Kantor, P.B., Lopresti, D.P., Zhou, J. (eds.) Proceedings of Document Recognition and Retrieval VIII(IS&T/SPIE Electronic Imaging), San Jose, CA, vol. 4307, pp. 44–55. (2001)
21. W3, HTML: The Markup Language (an HTML language reference). Retrieved 25 Sept 2015. http://www.w3.org/TR/html-markup/syntax.html#doctype-syntax
22. Creativyst, The Comma Separated Value (CSV) File Forma. http://creativyst.com/Doc/Articles/CSV/CSV01.htm
23. Gatterbauer, W., Bohunsky, P., Krüpl, B., Pollak, B., Herzog, M.: Towards Domain Independent Information Extraction from Web Tables. In: WWW, Banff, Alberta, Canada, 8–12 May 2007
24. Amano, A., Asada, N.: Graph grammar based analysis system of complex table form document. In: Proceedings of the Seventh International Conference on Document Analysis and Recognition (2003)
25. Bing, L., Zao, J., Hong, X.: New method for logical structure extraction of form document image. In: Proceedings of Document Recognition and Retrieval VI (IS&T/SPIE Electronic Imaging '99), San Jose, CA, vol. 3651, pp. 183–193 (1999)
26. Kieninger, T., Dengel, A.: A paper-to-HTML table converting system. In: Proceedings of Document Analysis Systems, (DAS) 98, Nagano, Japan (1998)
27. Coüasnon, B., Camillerapp, J., Leplumey, I.: Making handwritten archives documents accessible to public with a generic system of document image analysis. In: Proceedings of the International Workshop on Document Image Analysis for Libraries, Palo Alto, CA, pp. 270–277 (2004)
28. Martinat, I., Coüasnon, B., Camillerapp, J.: An adaptative recognition system using a table description language for hierarchical table structures in archival documents. In: Graphics Recognition: Recent Advances and Perspectives. Lecture Note in Computer Science, vol. 5046, pp. 9–20. Springer (2008)

29. Lemaitre, A., Camillerapp, J., Coüasnon, B.: Multiresolution cooperation improves document structure recognition. Int. J. Doc. Anal. Recognit. (IJDAR) **11**(2), 97–109 (2008)
30. Klein, B., Agne, S., Dengel, A.: On benchmarking of invoice analysis systems. In: Bunke, H., Spitz, A.L. (eds.) DAS 2006, LNCS, vol 3872, pp 312–323. Springer, Heidelberg (2006)
31. Klein, B., Dengel, A.: Problem-adaptable document analysis and understanding for high-volume applications. IJDAR **6**(3), 167–180 (2003)
32. Hamza, H., Belaid, Y., Belaid, A.: A case-based reasoning approach for invoice structure extraction. In: Proceedings of the Ninth International Conference on Document Analysis and Recognition, ICDAR 2007, vol. 1, pp. 327–331 (2007)
33. Watanabe, T., Quo, Q.L., Sugie, N.: Layout recognition of multikinds of table-form documents. IEEE Trans. Pattern Anal. Mach. Intell. **17**(4), 432–445 (1995)
34. Shamalian, H., Baird, H.S., Wood, T.L.: Are targetable table reader. In: Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'97), pp. 158–163 (1997)
35. Fang, J., Mitra, P., Tang, Z., Giles, L.: Table header detection and classification. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, vol. 599–605 (2012)
36. Shigarov, A.O.: Table understanding using a rule engine. Expert Syst. Appl. **42**(2), 929–937 (2015)
37. Halevy, A., Norvig, P., Pereira, F.: The unreasonable effectiveness of data. In: IEEE Intelligent Systems (2009)
38. Venetis, P., Halevy, A., Madhavan, J., Pasca, M., Shen, W., Wu, F., Miao, G., Wu, C.: Recovering semantics of tables on the web. In: Proceedings of the LDB Endowment, vol. 4, 9th ed. (2011)
39. Gonzalez, H., Halevy, A.Y., Jensen, C.S., Langen, A., Madhavan, J., Shapley, R., Shen, W., Goldberg-Kidony, J.: Google fusion tables: web-centered data management and collaboration. In: SIGMOD' 10, Indianapolis, Indiana, USA, 6–11 June 2010
40. Adelfio, M.D., Samet, H.: Schema extraction for tabular data on the web. In: Proceedings of The 39th International Conference on Very Large Data Bases, (Proceedings of the VLDB Endowment, vol. 6, 6th ed.), Riva del Garda, Trento, Italy 26–30 August 2013
41. Long, V.: An agent-based approach to table recognition and interpretation, Macquarie University Ph.D. dissertation, May (2010)
42. Astrakhantsev, N.: Extracting objects and their attributes from tables in text documents. In: Turdakov, D., Simanovsky, A. (eds.) Proceedings of the Seventh Spring Researchers Colloquium on Databases and Information Systems, SYRCoDIS 2011, Moscow, Russia, CEUR Workshop Proceedings 735 CEUR-WS. org 2011 pp. 34–37 (2011)
43. Hurst, M., Douglas, S.L: Layout and language: preliminary investigations in recognizing the structure of tables. In: Proceedings of the International Conference on Document Analysis and Recognition (ICDAR'97), pp. 1043–047 (1997)

44. Hurst, M.: Towards a theory of tables. Int. J. Doc. Anal. Recognit. **8**(2–3), 66–86 (2006). (Springer, Heidelberg)
45. Hurst, M.: The interpretation of tables in texts, Ph.D. thesis, University of Edinburgh, (2000)
46. Costa e Silva, A., 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)
47. Kim, Y.-S., Lee, K.-Y.: Extracting logical structures from HTML tables. Comput. Stand. Interfaces **30**(5), 296–308 (2008)
48. Pivk, A., et al.: Transforming arbitrary tables into logical form with TARTAR. Data Knowl. Eng. **60**, 567–595 (2007)
49. Chen, Z., Cafarella, M.: Automatic web spreadsheet data extraction. In: Proceedings of the 3rd International Workshop on Semantic Search over the Web (SSW 2013), Riva del Garda, Trento, Italy, 30 Aug (2013)
50. Astrakev, N., Turdakov, D., Vassilieva, N.: Semi-automatic data extraction from tables. In: Proceedings of the 15th All-Russian Conference on Digital Libraries: Advanced Methods and Technologies, Digital Collection—RCDL, Yaroslavl, Russia (2013)
51. Kasar, T., Bhowmik, T.K., Belaid, A.: Table information extraction and structure recognition using query patterns. In: Proceedings 13th International Conference on Document Analysis and Recognition, ICDAR 2015, vol. 1, pp. 1086–1080 (2015)
52. Lopresti, D., Nagy, G.: Automated table processing: an (opinionated) survey. In: Proceedings of IAPR Workshop on Graphics Recognition (GREC99), Jaipur, India, pp. 109–134, Sept (1999)
53. Hu, J., Kashi, R., Lopresti, D., Wilfong, G., Nagy, G.: Why table ground-truthing is hard. In: Proceedings of International Conference on Document Analysis and Recognition, pp. 129–133. IEEE Computer Society Press, Seattle, WA, Sept (2001)
54. Embley, D.W., Lopresti, D., Nagy, G.: Notes on contemporary table recognition. In: Bunke, H., Spitz, A.L., (eds.) Proceedings of the 7th International Workshop on Document Analysis Systems VII DAS 2006, vol. 3872, LNCS, pp. 164–175, Springer, Nelson, New Zealand, 13–15 Feb (2006)
55. Embley, D.W., Lopresti, D., Hurst, M., Nagy, G.: Table processing paradigms: a research survey. In: International Journal of Document Analysis and Recognition, vol. 8, 2–3rd ed., pp. 66–86. Springer, June (2006)
56. Embley, D., Tao, C., Liddle, S.: Automating the extraction of data from HTML tables with unknown structure. Data Knowl. Eng. **54**(1), 3–28 (2005)
57. Tao, C., Embley, D.W.: Automatic hidden-web table interpretation, conceptualization, and semantic annotation. Data Knowl. Eng. **68**(7), 683–703 (2009)
58. Jandhyala, R.C., Krishnamoorthy, M., Nagy, G., Padmanabhan, R., Seth, S., Silversmith, W.: From tessellations to table interpretation. In: Carette, J. et al. (eds.) Proceedings of the 8th International Conference on Mathematical Knowledge Management, MKM 2009, Grand Bend, Ontario, Calculemus/MKM 2009, LNAI 5625, pp. 422–437. Springer, Berlin (2009)

59. Nagy, G.: Learning the characteristics of critical cells from web tables. In: Proceedings of the ICPR, Tsukuba, Japan, Nov (2012)
60. Embley, D.W., Krishnamoorthy, M., Nagy, G., Seth, S.: Factoring Web Tables. In: Mehrotra, K.G. et al. (eds.): IEA/AIE 2011, Part I, LNAI 6703, pp. 253–263. Springer, Berlin (2011)
61. Nagy, G., Tamhankar, M.: VeriClick, an efficient tool for table format verification. In: Proceedings of the SPIE 8297, Document Recognition and Retrieval XIX, 82970M, 23 Jan 2012
62. Seth, S., Nagy, G.: Segmenting Tables via indexing of value cells by table headers. In: Proceedings of the ICDAR 2013,Washington, DC, Aug (2013)
63. Nagy, G., Embley, D.W., Seth, S.: End-to-end conversion of HTML tables for populating a relational database. In: Proceedings of the DAS 2014, Tours, France (2014)
64. Embley, D.W., Seth, S., Nagy, G. : Transforming Web tables to a relational database. In: Proceedings of the ICPR 2014, Stockholm, Sweden (2014)
65. Embley, D.W., Seth, S., Krishnamoorthy, M., Nagy, G.: Clustering header categories extracted from web tables. In: Proceedings SPIE/IST Document Recognition and Retrieval, San Francisco, CA, Feb (2015)
66. U.S. Government Printing Office, Style Manual: An official guide to the form and style of Federal Government printing, section 13, 281–299. http://www.gpoaccess.gov/stylemanual/index.html (2008)
67. Balbiani, P., Condotta, J.-F., Farinas Del Cero, L.: Tractability results in the block algebra. J. Logic Comput. **12**(5), 885–909 (2002)
68. Allen, J.F.:Maintaining knowledge about temporal intervals. Commun. ACM **26**(11), 832–843 (1983)
69. Padmanabhan, R., Jandhyala, R.C., Krishnamoorthy, M., Nagy, G., Seth, S., Silversmith, W.: Interactive conversion of large web tables. GREC **25–36**, 2009 (2009)
70. Cafarella, M.: http://web.eecs.umich.edu/~michjc/structuredweb/index.html (Accessed 6 Jan 2016)
71. W3C Semantic Web: Resource Description Framework (RDF). Retrieved 1/31/2015 from https://www.w3.org/RDF/ (2014)
72. W3C Semantic Web: Web Ontology Language (OWL). Retrieved 1/31/2015 from https://www.w3.org/OWL (2013)