Rule-Based Table Analysis and Interpretation

Alexey Shigarov(✉)

Matrosov Institute for System Dynamics and Control Theory SB RAS,
Lermontov Str. 134, 664033 Irkutsk, Russia
shigarov@icc.ru

Abstract. Today, a huge amount of tables are presented in web pages, word documents, and spreadsheets. Many of them are unstructured tabular data. They are intended to be understood by humans but not to be interpreted by machines. At the same time, we often need to have that information in a structured form, e.g. relational databases. We propose a rule-based approach to table analysis and interpretation and demonstrate how it can be applied to transform tabular data from unstructured (spreadsheets) to structured (relational databases) form. The paper discusses representing tabular data as facts in the working memory of a rule engine, a formal language for defining rules of table analysis and interpretation, and its implementation.

Keywords: Table analysis and interpretation · Table understanding · Information extraction from tables · Unstructured tabular data integration

1 Introduction

Today, a huge amount of tables are presented in web pages, word documents, and spreadsheets. Many of them are unstructured tabular data. It refers to any tabular information, which is not organized as a table of a relational database. These tables are intended to be interpreted by humans but not designed for high-level machine processing like SQL queries.

In practice, the transformation of tabular data from unstructured to structured form is required in many cases. For example, tables presented in unstructured form are often the only available source of statistical or financial information. To use that information in business intelligence we need to transform data from these tables to structured form like relational databases.

The conversion of unstructured tabular data to structured form can be considered as table understanding [1,2], which consists in recovering relationships among entries (data values), labels (attributes), and categories (dimensions) presented in a table. Our work is devoted to the issues of recovering semantic relationships of a table. In terms of Hurst [1], we deal with the following steps of table understanding: functional analysis separating cells into entries and labels, structural analysis recovering relationships between cells, and interpretation recovering relationships between entries, labels, and categories.

G. Dregvaite and R. Damasevicius (Eds.): ICIST 2015, CCIS 538, pp. 175–186, 2015.
DOI: 10.1007/978-3-319-24770-0_16

There are several challenges in the table understanding. A table can be produced or generated by a huge amount of ways. Table features originate from typographical standards, corporative practice, ad hoc software, data formats, and human inventiveness. To reduce the complexity of table understanding the existing methods use various assumptions (heuristics) about tables. Usually those assumptions are entirely embedded in their algorithms. This constrains a range of tables, which can successfully be understood by them.

We propose a rule-based approach for transforming unstructured tabular data to relational databases [3]. The main idea we exploit is that tables produced by the same vendor often have similar structures, styles, and content. It allows defining a set of production rules for describing how these tables can be analyzed and interpreted. We propose to develop separate sets of table interpretation rules (knowledge bases) for different sets of similar tables. In that case, the process of the table analysis and interpretation is performed as rule firing. It provides processing of a wide range of tables having various complex structures and features.

Based on the approach we develop a table model for presenting tabular data as facts in the working memory of a rule engine and a formal language for defining rules of table analysis and interpretation, called CRL (Cells Rule Language). These rules map what we know, i.e. spatial (topological), style (typographical), and textual (natural language) information of a table, into what we do not know, i.e. its semantic relationships (label-entry, label-label, and label-category pairs). Our language is implemented as domain specific language for the rule engine "Drools Expert" [4]. It allows translating CRL rules to DRL (Drools Rule Language) [4] and firing them in this rule engine. The concepts of CRL language are examined with our prototype of the system for conversion of unstructured tabular data from spreadsheets to structured (canonical) form.

The remainder of this paper is organized as follows. In Sect. 2 we discuss the studies devoted to the issues of table analysis and interpretation. Section 3 describes our data structures for representing tabular data as facts in the working memory of the rule engine. The CRL language is considered in Sect. 4. In Sect. 5 we demonstrate several typical and complex table structures, and how they can be analyzed and interpreted by CRL rules.

2 Related Work

Existing methods for table analysis and interpretation can be divided into two groups: domain-specific [5–7] and domain-independent [9–14].

The domain-specific methods are based on using ontologies or knowledge bases describing a particular domain. These methods allow binding natural language content of a table with concepts of the particular domain. For instance, the method from the TANGO project [5] is based on a library of frames containing knowledge about lexical content of tables. Each frame describes a data type using regular expressions, dictionaries, and open resources like the lexical database WordNet. Embley et al. [6] use ontologies developed specially for

information extraction. In addition to objects, relationships and constraints the extraction ontology includes a set of data frames, which are associated with sets of objects. Those data frames allow binding table content with objects of the ontology using regular expressions. Wang et al. [7] consider the problem of understanding a web table as associating the table with semantic concepts presented in the "Probase" [8] knowledge base.

The methods listed above [5–7] principally use the analysis of natural language content from tables. It is not always sufficient in practice. Information extraction from tables often requires the analysis of spatial and style information for high accuracy.

The domain-independent methods [9–14] are based on the analysis and interpretation of spatial, style and text information from tables instead of using external knowledge on a specific domain. For instance, Gatterbauer et al. [9] propose to use only the analysis of spatial and style information in CSS2 format. Their method is based on assumptions about style information designed for several common types of web-tables. Also Pivk et al. [10,11] suggest the methodology and TARTAR system for automatic transforming HTML tables into logical structured form (semantic frames). The TARTAR system uses heuristics on structure and text content of a table, which are designed for three typical table types. Kim et al. [12] use the analysis of spatial, style, and natural language information from web tables based on embedded rules and regular expressions for five table types. The recent papers [13,14] discuss the method for transforming data from web tables to a relational database. The method provides grouping attributes into categories, using only the analysis of table structure. It is based on several embedded in algorithms assumptions on regular structure of pivot tables.

The mentioned above domain-independent methods [9–14] are based on using a limited set of assumptions on table structures, styles, and content which originate from a few common types of tables. These assumptions are embedded in the proposed algorithms. They limit classes of tables, which can be analyzed and interpreted by those methods with high accuracy.

3 Tabular Data Facts

To design data structures representing table facts, we are inspired by the Wang's table model [15]. We use partially terminology of [15] to describe a table: entries (data values), labels (attributes), and categories (dimensions). These concepts and their relationships are shown in Fig. 1.

Cell is a main structure for representing input tabular data facts extracted from a cell of a table. Each cell characteristic is accessible through the corresponding field. The cell structure includes the following main fields (hereafter, they are marked by monospaced font):

– Positions: `cl` — left column, `rt` — top row, `cr` — right column, and `rb` — bottom row. Note that a cell located on several consecutive rows and columns

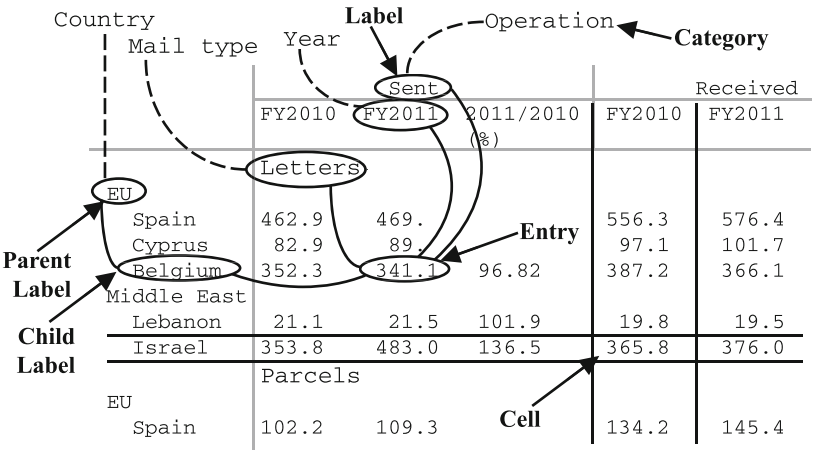

Fig. 1. Table concepts and their relationships

covers a few grid tiles, which always compose a rectangle. Moreover, two cells cannot overlap each other.

- Style settings `style` are encapsulated in the style structure. The main of them are: `font` contains all typical font characteristics; `horzAlignment` and `vertAlignment` indicate types of horizontal and vertical alignments respectively; `rotation` defines text rotation; `bgColor` and `fgColor` represent background and foreground colors; `leftBorder`, `topBorder`, `rightBorder`, and `bottomBorder` specify types (thin, medium, dashed, etc.) and colors of four corresponding cell borders.
- Content: a cell can contains text, image, or RTF. However, the current cell structure supports only textual content through the field `text`.
- There are several additional characteristics: `cellType` — data type (numeric, date, string, etc.), `indent` — number of space characters in the beginning of the textual content, `width` in units of 1/256th of a character width and `height` in twips (1/20th of a point), `mark` — word marking the cell, `table` — reference to the corresponding table structure, `entries` — ordered set of entries and `labels` — ordered set of labels generated from the cell. Note that, a cell can contain one or more entries and/or one or more label in the same time.

Entry structure serves to present data values from a table. Entry consists of the following fields: `value` — textual value, `cell` — reference to the related cell, `labels` — set of labels associated with the entry. Moreover, an entry can be associated with only one label in each category.

Label structure represents labels, which are values describing entries. A label is defined by the fields: `value` — textual value, `cell` — reference to the related cell if it exists. Also, each label is associated with only one category: `category` — reference to it. Labels of a category can be organized as one or more trees. Therefore, a label can have a parent and a set of children. They can be accessible through the fields: `parent` and `children`.

Category structure intends to represent categories (dimensions). One or more labels are combined into a category. This structure includes the following fields: `name` — textual value, `labels` — set of related labels.

4 CRL Rules

A CRL rule defines how we analyze and interpret a table. The left hand side of the rule defines conditions using known facts about a table. Its right hand side contains consequences (actions) to recover its unknown semantic relationships. The specification of the CRL language is defined as a set of DSL definitions, which map CRL sentences to DRL constructs. It is available at the following address: http://cells.icc.ru/pub/crl. The key CRL constructs are presented in the paper.

4.1 Conditions

The condition elements allow querying cells, entries, labels, and categories asserted as facts into the working memory:

```
cell $cell : constraints
entry $entry : constraints
label $label : constraints
category $category : constraints
```

A condition element consists of three parts. In its order of occurrence, the first is a keyword which denotes one of the following fact types: `cell`, `entry`, `label`, or `category`. The second is a variable name that starts with a dollar sign ('$'). The third optional part defines constraints restricting the requested facts. They follow the colon character (':'). A constraint is an expression that returns "true" or "false". These constraints conform to the DRL syntax. Therefore, in essence, they are Java expressions with some enhancements. Also, they can be separated by the comma character (',') that is logical conjunction for them. A condition element without constraints allows querying all facts of specified type.

4.2 Consequences

Cell Marking is an optional action which allows binding a cell `$cell` with a mark `@mark`, that is a word with the first character '@':

```
set mark @mark -> $cell
```

Using marks allows developing more understandable rules when it is possible to divide cells into several meaningful groups and apply different subsets of rules for them. In these cases, a constraint on a mark can substitute several constraints on other cell characteristics.

The typical practice is to set a mark to all cells, which play the same function or are located in the same table region. For example, if a table has the following parts: "head", "stub", and "body", we can mark each cell with one of the marks: `@head`, `@stub`, or `@body`, depending on their location in the table. Subsequently, we can use these marks in other rules, instead of using constraints on cell location in table regions.

Label and Entry Creating consequences generate entries and labels in a cell `$cell` respectively, using the string expressions `entry_value` and `label_value` usually extracted from its textual content:

```
new entry entry_value -> $cell
new label label_value -> $cell
```

Each created entry or label is asserted into the working memory as a new fact. Moreover, the following short form can be used to set entry and label values respectively, using a cell text without string processing:

```
new label $cell
new entry $cell
```

Label Categorizing is to associate a label `$label` with a category `$category`:

```
set category $category -> $label
```

Furthermore, a string expression `category_name` presenting the name of a category can be used as the argument:

```
set category category_name -> $label
```

In the latter case, we try to find the category with this name in the current table instance. If it exists, then the label is associated with it. Otherwise, we try to create locally in the table the new category with this name and then associate the label with it.

Label-Label Associating allows connecting two labels `$label1` and `$label2`:

```
set parent label $label1 -> $label2
```

In the consequence above, the argument `label1` is considered as the parent and the addressee `$label2` respectively is its child. This action mainly intends to support hierarchical categories. Two or more labels can be connected, organizing a tree. An attempt of creating a cycle in label-label relationships causes that the table processing cannot perform the further rule firing. Additionally, we suppose that all labels connected in the tree must be associated with the same category. The contrary case brings the table processing to a halt.

Label Grouping places two labels `$label1` and `$label2` in one group:

```
group $label1 -> $label2
```

A group is a set of labels, that can be considered as an anonymous category. In some tables, we can define that several labels are related to the same category, without knowing what the category is. For example, we may know that labels, which are located in the same row, are related to the same category. Placing two or more labels in the group means that they are related to the same category.

In time of the rule firing, one or more labels from a group can be associated with a category. When it happens, we assume that all the rest labels from the group also must be associated with the category. After the rule firing, we try to associate these labels with the category. The case, when two labels are grouped together but associated with different categories, is not allowed and leads to the crash of the table processing. After the rule firing, if in a group all labels are not categorized, then the category with automatically generated name is created and these labels are associated with it.

Entry-Label Associating is used to relate an entry `$entry` with a label `$label`:

```
add label $label -> $entry
```

As is mentioned above, any entry can be associated with only one label from each category. The interruption of this is considered as the failure that does not allow the further table processing. Moreover, it means that the label must belong to a category. If the added label is uncategorized then it is not associated with the entry at that moment. At first, it becomes a candidate which may be associated automatically with the entry only after it is categorized.

There are two additional forms for the consequence. The first serves to associate an entry `$entry` with a label having the value specified by the first argument `label_value` from a category with the name defined by the second argument `category_name` (both of them are string expressions):

```
add label label_value from category_name -> $entry
```

First, to process this consequence, we examine if the current table instance has the category with this name. If it does not exist, we try to create this category. After that, we look for the label with the specified value in the founded or created category. When there is no label, we create it, using this value. At last, the entry is associated with the founded or created label. Note that, this form allows to generate labels independently of cells.

The second form is similar to the first, but the second argument `$category` is a variable that refers to a category:

```
add label label_value from $category -> $entry
```

In this case, we try to find or to create the label specified with the value `label_value` in the category `$category`, and then to relate the entry `$entry` to it.

Auxiliary Consequences. There are several auxiliary consequences, including the following: cell splitting and merging; editing textual content of a cell; editing a value of an entry or label; and updating facts such as cells, entries, and labels in the working memory.

5 Applying CRL

We demonstrate several typical and complex table structures, and how they can be analyzed and interpreted by CRL rules. More examples of CRL rules can be found at the address http://cells.icc.ru/pub/crl/samples.

5.1 Marking, Creating, Grouping, and Categorizing

In tables like the ones shown in Fig. 2, *a*, cells can be separated into three region: column headings, row headings, and data values. Actually, two labels, where one is produced from a column heading and other is generated from a row heading, belong to different categories. We can use the empty cell in the top-left corner to allocate cells among these regions. For example, the rule below can be used to mark column headings (any cell `$cell` located on the right of `$corner`, but in the same rows) and to create corresponding labels:

```
when
    cell $corner : cl == 1, rt == 1, blank
    cell $cell : cl > $corner.cr, rb <= $corner.rb
then
    set mark @ColumnHeading -> $cell
    new label $cell
```

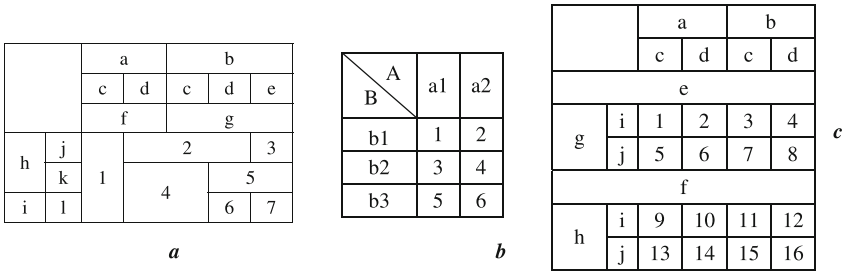

Fig. 2. Pivot tables: the table contains the data values indicated as numbers ('1',...,'7'), column ('a',...,'g') and row ('h',...,'l') headings (*a*) presented by Latin characters for convenience; the cell in the top-left corner contains two headings 'A' and 'B', describing the headings in the boxhead ('a1','a2') and stub ('b1','b2','b3') respectively (*b*); and the cut-in headings 'e' and 'f' appear between data cells (*c*)

Furthermore, we suppose that two labels, which are generated from either two column headings located in one row or two row headings situated in one column, are connected with the same category. So, the labels in Fig. 2, *a* belong to the five categories as follows: {'a', 'b'}, {'c', 'd', 'e'}, {'f', 'g'}, {'h', 'i'}, and {'j', 'k', 'l'}. Using this assumption, labels can be grouped and associated with corresponding anonymous categories. For example, the following rule allows grouping the labels related to column headings:

```
when
    label $l1 : cell.mark =="@ColumnHeading", $c : cell
    label $l2 : cell.(mark == "@ColumnHeading", rt == $c.rt)
then
    group $l1 -> $l2
```

For tables like Fig. 2, *b*, we assume that the top-left corner contains headings (like 'A' and 'B'), which define two category names: the first ('A') for labels created from boxhead cells, and the second ('B') for labels originated from the stub cells. We can use them for categorizing labels. For example, the rule, where we set a category to column headings, can be written as follows:

```
when
    cell $corner : cl == 1, rt == 1, $t : text
    label $label : cell.rb <= $corner.rb
then
    set category token($t, 0) -> $label
```

Here, the function `token` returns the first ('0') token of a text '$t' of the top-left corner cell `$corner`.

In tables like Fig. 2, *c*, supposing that a cut-in heading cell spans all columns, we can write the following rule for marking these cells:

```
when
    cell $cell : cl == 1, cr == table.numOfCols
then
    set mark @CutInHeading -> $cell
```

In this rule, the nested field `table.numOfCols` is a number of columns of the table.

5.2 Processing Multi-valued Cells and Footnotes

In a bilingual table as Fig. 3, *a*, a cell can contain two labels or two entries in both languages. For example, the rule for generating labels from cells located in the leftmost column or the topmost row, where the first phrase is written in Greek language and the second is Chinese phrase, may have the form:

```
when
    cell $cell : cl == 1 || rt == 1, !blank, $t : text
then
    new label extract($t, "\\p{IsGreek}+") -> $cell
    new label extract($t, "\\p{IsHan}+") -> $cell
```

Here, the function `extract` returns all occurrences of a text `$t` which are matched to the regular expression `"\p{IsGreek}+"` Greek and `"\p{IsHan}+"` for Chinese language respectively.

In this example, we suppose that the first entry in a cell can be related exclusively to the first label in other cell, as well as the second entry only to the second label. The rule implementing this assumption is shown below:

```
when
    cell $c1 : containsLabel()
    cell $c2 : containsEntry(), cl == $c1.cl || rt == $c1.rt
```

```
then
    add label $c1.label[0] -> $c2.entry[0]
    add label $c1.label[1] -> $c2.entry[1]
```

For tables similar to the one shown in Fig. 3, *b*, where any cell under the topmost row contains a text as "key=value", the following rule creates a label from the "key" part and an entry from the "value" part:

```
when
    cell $cell : rt > 1, $t : text
then
    new label left($t,'=') -> $cell
    new entry right($t,'=') -> $cell
```

In the first consequence above, the function `left` returns a substring of a text `$t` before the '=' character. In the second, the function `right` returns a substring after this character.

Recovering relationships between a label ("key") and an entry ("value") can be realized as follows:

```
when
    cell $cell : rt > 1
then
    add label $cell.label -> $cell.entry
```

Footnotes can be interpreted differently, depending on the requirements of target representation. In our example (Fig. 3*c*), the footnotes ('u' and 'v'), which are related to the entries ('2' and '5') through the references ('*', '**') respectively, are considered as labels. The following rule shows how to create a label from a footnote and relate it to a corresponding entry.

```
when
    cell $footer : rb == table.numOfRows, $footnotes : text
    entry $entry : cell.text matches ".+\\*+",
        $reference : extract(cell.text, "\\*+")
then
    add label between($footnotes, $reference, '\n')
        from "Footnote" -> $entry
```

In the first condition above, we query a text `$footnotes` of the cell `$footer` located on the bottommost row. The second condition: we try to find all cells having a text ending by one or more asterisk ('*') character. The text `$reference` corresponds the footnote reference extracted from the text by the regular expression `"\*+"`. In the consequence, the function `between` returns a substring of the text `$footnotes` between the reference `$reference` and the newline character. We create a label in the category named "Footnote", using the substring as its value, and associate the entry `$entry` with it.

	α 阿爾法	β 公測
γ 伽馬	1 一	2 二
δ 三角洲	3 三	4 四

a

C1	C2
a = 1	b = 2
c = 3	d = 4
e = 7	f = 8
g = 10	h = 11

b

	a	b	c
d	1	2*	3
e	4	5**	6
f	7	8	9
* u ** v			

c

Fig. 3. Tables with multi-valued cells: the bilingual table, where each non-empty cell has either two labels or two entries (*a*); a text like "key=value" in a cell can be interpreted as a label ("key" part) and a related entry ("value" part) (*b*); footnotes 'u' and 'v' in the footer

6 Conclusions

Our rule-based approach to table analysis and interpretation is implemented in the CRL language. CRL rules can be translated to the DRL format and executed by the "Drools Expert" rule engine.

As in existing methods, we also use assumptions about structures, styles and content of tables. But, in contrast to them, we divide assumptions into two parts: general and special. General assumptions are embedded in our data structures described in Sect. 3. Special assumptions are written with the CRL language. They are combined into sets (knowledge bases), which are designed for different classes of tables. That approach allows reaching high or even absolute accuracy for particular classes of tables.

Furthermore, the CRL rules provide possibilities of dealing with not typical table features, including the following: headings and data cells can be located anywhere (e.g. footers, cut-in heads), non-numerical data values; multi-valued cells (several entries and/or labels can be placed in a cell); hierarchy of labels built by indents in text; footnotes.

The CRL language can be used in developing software for unstructured tabular data integration, populating databases from spreadsheets, and extracting information from tables.

Acknowledgments. The research work was financially supported by the Russian Foundation for Basic Research (Grant No. 15-37-20042) and the Council for grants of the President of the Russian Federation (Grant No. SP-3387.2013.5).

References

1. Hurst, M.: Layout and language: challenges for table understanding on the web. In: 1st International Workshop on Web Document Analysis, pp. 27–30, Seattle (2001)
2. Embley, D.W., Hurst, M., Lopresti, D., Nagy, G.: Table-processing paradigms: a research survey. IJDAR **8**(2), 66–86 (2006)
3. Shigarov, A.O.: Table understanding using a rule engine. Expert Syst. Appl. **42**(2), 929–937 (2015)

4. Drools Expert. http://www.drools.org
5. Tijerino, Y.A., Embley, D.W., Lonsdale, D.W., Ding, Y., Nagy, G.: Towards ontology generation from tables. World Wide Web Internet Web Inf. Syst. **8**(3), 261–285 (2005)
6. Embley, D.W., Tao, C., Liddle, S.W.: Automating the extraction of data from HTML tables with unknown structure. Data Knowl. Eng. **54**(1), 3–28 (2005)
7. Wang, J., Wang, H., Wang, Z., Zhu, K.Q.: Understanding tables on the web. In: Atzeni, P., Cheung, D., Ram, S. (eds.) ER 2012 Main Conference 2012. LNCS, vol. 7532, pp. 141–155. Springer, Heidelberg (2012)
8. Probase. http://research.microsoft.com/en-us/projects/probase
9. Gatterbauer, W., Bohunsky, P., Herzog, M., Krpl, B., Pollak, B.: Towards domain-independent information extraction from web tables. In: 16th International Conference on World Wide Web, pp. 71–80. ACM, Banff (2007)
10. Pivk, A., Cimianob, P., Sure, Y.: From tables to frames. Web Seman. Sci. Serv. Agents World Wide Web. **3**(2–3), 132–146 (2005)
11. Pivk, A., Cimiano, P., Sure, Y., Gams, M., Rajkovic, V., Studer, R.: Transforming arbitrary tables into logical form with TARTAR. Data Knowl. Eng. **60**(3), 567–595 (2007)
12. Kim, Y.-S., Lee, K.-H.: Extracting logical structures from HTML tables. Comput. Stan. Interfaces **30**(5), 296–308 (2008)
13. Embley, D.W., Nagy, G., Seth, S.: Transforming web tables to a relational database. In: 22nd International Conference on Pattern Recognition, pp. 2781–2786. IEEE Computer Society, Washington (2014)
14. Nagy, G., Embley, D.W., Seth, S.: End-to-end conversion of HTML tables for populating a relational database. In: 11th IAPR International Workshop on Document Analysis Systems, pp. 222–226. IEEE Computer Society, Troy (2014)
15. Wang, X.: Tabular Abstraction, Editing, and Formatting. PhD Thesis. University of Waterloo, Waterloo (1996)