Rule-Based Spreadsheet Data Transformation from Arbitrary to Relational Tables

Alexey O. Shigarov[a,*], Andrey A. Mikhailov[a]

[a] *Matrosov Institute for System Dynamics and Control Theory of SB RAS, 134 Lermontov st., Irkutsk, Russia, 664033*

Abstract

The paper discusses issues of rule-based data transformation from arbitrary spreadsheet tables to a canonical (relational) form. We present a novel table object model and rule-based language for table analysis and interpretation. The model is intended to represent a physical (cellular) and logical (semantic) structure of an arbitrary table in the transformation process. The language allows drawing up this process as consecutive steps of table understanding, i. e. recovering implicit semantics. Both are implemented in our tool for spreadsheet data canonicalization. The presented case study demonstrates the use of the tool for developing a task-specific rule-set to convert data from arbitrary tables of the same genre (government statistical websites) to flat file databases. The performance evaluation confirms the applicability of the implemented rule-set in accomplishing the stated objectives of the application.

Keywords: spreadsheet data transformation, table understanding, table model, table analysis, table interpretation, rule-based programming

1. Introduction

Spreadsheets provide a popular way for creating and circulating arbitrary tables (e. g. cross-tabulations, invoices, roadmaps, and data collection forms).

*Corresponding author

Email addresses: shigarov@icc.ru (Alexey O. Shigarov), mikhailov@icc.ru (Andrey A. Mikhailov)

They can be considered as a general form for representing tabular data with an explicitly presented layout (cellular structure) and style (graphical formatting). For example, HTML tables presented in web pages can be easily converted to spreadsheet formats. The arbitrary tables can be a valuable data source in business intelligence and data-driven research. However, difficulties that inevitably arise with extraction and integration of the tabular data often hinder the intensive use of them in the mentioned areas.

The number of genuine tables in the Web reaches hundreds of millions (Cafarella et al., 2008; Eberius et al., 2015; Lehmberg et al., 2016). Many of them are relational tables that can be considered as flat databases. Nevertheless, there are other popular types of tables (Crestan & Pantel, 2011; Chen & Cafarella, 2013; Lautert et al., 2013; Braunschweig, 2015) having layout features designed for human understanding (e. g. merged cells, footnotes, and indentations). These include about 50% of tables presented in 0.4M spreadsheets of ClueWeb09 Crawl[1] (Chen & Cafarella, 2013) and 147M (61%) of 233M web tables extracted from Common Crawl[2] (Lehmberg et al., 2016). They lack explicit semantics required for computer programs to interpret their layout and content.

Table understanding is to recover the missing semantics. The papers (Hurst, 2001; e Silva et al., 2006) defines the five consecutive stages of the table understanding: *detection* of a table in a document, *recognition* (segmentation) of its cellular structure, *functional* and *structural analysis* for recovering its logical structure, and *interpretation* that aspire to recover its semantics through linking its content with target schema or domain concepts.

We regard the transformation of spreadsheet tabular data (Fig. 1, *a*) into the relational form (Fig. 1, *b*) as a process of table understanding. In the general case, this transformation includes all the enumerated stages:

1. *Detection.* A spreadsheet document can contain several arbitrary tables

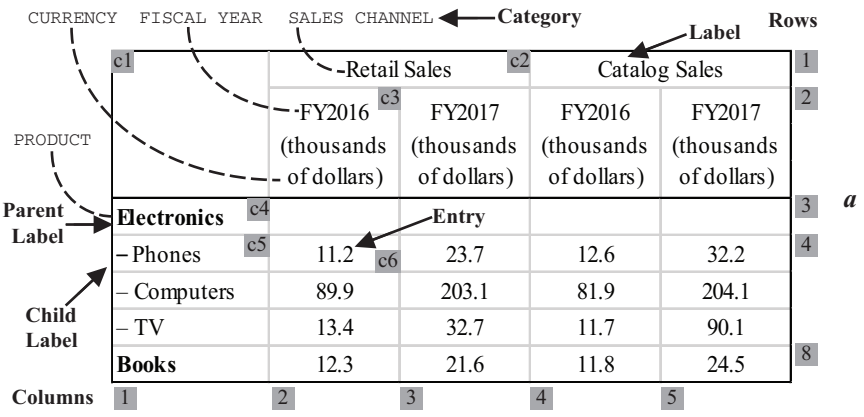
[1] `http://lemurproject.org/clueweb09`
[2] `http://commoncrawl.org`

surrounded by text and graphics.

2. *Recognition.* A human-readable structure of an arbitrary table can differ from its machine-readable structure presented in a spreadsheet, e. g. one logical cell can be visually composed of several physical cells through drawing their borders.
3. *Role (functional) analysis.* A spreadsheet cell stores a text, where a human can distinguish one or more data items that play some functional roles in a table (e. g. values or attributes). However, there are no spreadsheet metadata that separate data items from a cell value and determine their functional roles.
4. *Structural analysis.* A spreadsheet also contains no metadata for representing relationships between data items of a table.
5. *Interpretation.* A data item can be an instance of a concept (category), but its spreadsheet does not explicitly associate it with a domain ontology or a global taxonomy.

The paper covers the rule-based analysis and interpretation of arbitrary tables presented in spreadsheets. Our contribution consists of the following results:

1. We present a novel table object model designed for representing a physical (cellular) and logical (semantic) structure of an arbitrary table in the transformation process (Section 2). Our model associates roles with data items instead of cells or cell regions (e. g. head, stub, or body). Moreover, it provides data provenance for recovered semantics.
2. We propose CRL (Cells Rule Language), a domain-specific language for expressing table analysis and interpretation rules (Section 3). A set of the rules can be implemented for a specific task characterized by requirements for source and target data.
3. We develope TabbyXL, a tool for rule-based transformation of arbitrary tables presented in spreadsheets into the canonical (relational) form (Section 4). The tool implements our table object model and rule language.



DATA	SALES CHANNEL	FISCAL YEAR	CURRENCY	PRODUCT
11200	retail	2016	u.s. dollars	electronics/phones
23700	retail	2017	u.s. dollars	electronics/phones
12600	catalog	2016	u.s. dollars	electronics/phones
32200	catalog	2017	u.s. dollars	electronics/phones

Figure 1: A fragment of a source arbitrary table (a); a fragment of a target table in the canonical form generated from the source table (b).

4. We evaluate an experimental application that is intended to convert data from tables of the same genre (government statistical websites) to flat file databases (Section 5). It exemplifies the use of our language for developing a task-specific rule-set. The performance evaluation confirms the applicability of the implemented rule-set in accomplishing the stated objectives of this application.

2. Table Object Model

The table object model is designed for representing both a physical structure and logical data items of an arbitrary table in the process of its analysis and interpretation (Fig. 2). Our model adopts the terminology of Wang's table model (Wang, 1996). It includes two interrelated layers: *physical* (Section 2.1) represented by the collection of cells and *logical* (Section 2.2) that consists of three collections of entries (values), labels (keys), and categories (concepts).

We deliberately resort to the two-way references between the layers to provide convenient access to their objects in table analysis and interpretation rules.

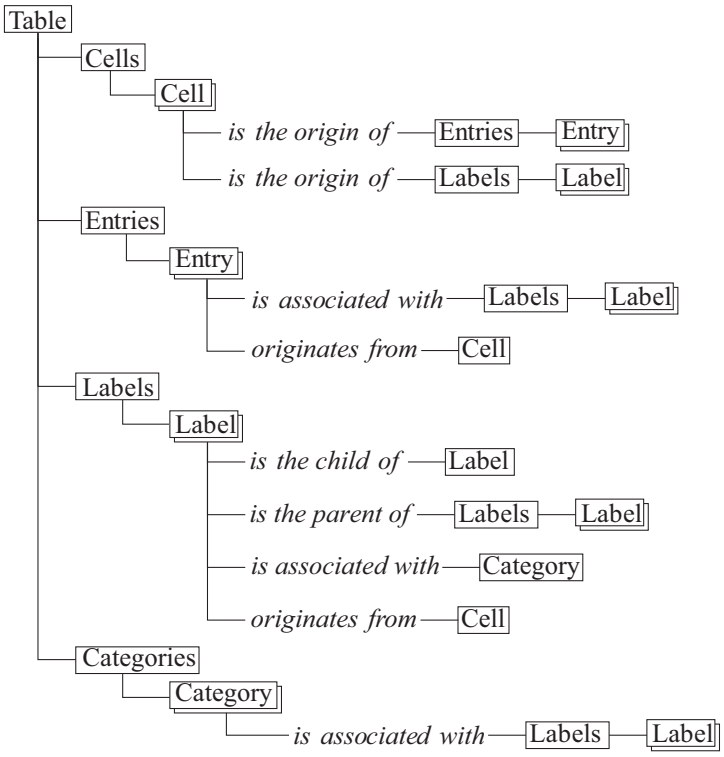


Figure 2: Two-layered table object model.

2.1. Physical Layer

`Cell` object models common features of a cell that can be presented in tagged documents of well-known formats, such as Excel, Word, or HTML. We define `Cell` object as a set of the following attributes:

- *Location*: `cl` — left column, `rt` — top row, `cr` — right column, and `rb` — bottom row. A cell located on several consecutive rows and columns covers a few grid tiles, which always compose a rectangle. Moreover, two cells cannot overlap each other.

- *Style*: `font` — font features (`name`, `color`, `height`, etc.), `bgColor` and `fgColor` — background and foreground colors, `rotation` — text rotation, `horzAlignment` and `vertAlignment` — horizontal and vertical alignment, `leftBorder`, `topBorder`, `rightBorder`, and `bottomBorder` — border features (types, colors), and `style` encapsulates the listed attributes.
- *Content*: `text` — textual content, `indent` — indent, and `type` — its literal data type (`NUMERIC`, `DATE`, `STRING`, etc.).
- *Annotation*: `mark` — a user-defined word or phrase to annotate the cell.
- *Logical layer references*: `entries` (a set of entries) and `labels` (a set of labels) originated from this cell. Thus, a cell can contain several entries and labels.

For example, an initial state of some cells (`c1`,...,`c6`) shown in Fig. 1, *a* can be represented as follows:

```
c1=(cl=1,rt=1,cr=1,rb=2,text=null)
c2=(cl=2,rt=1,cr=3,rb=1,text="Retail Sales")
c3=(cl=2,rt=2,cr=2,rb=2,text="FY2016 (thousands of dollars)")
c4=(cl=1,rt=3,cr=1,rb=3,style.font.bold=true,text="Electronics")
c5=(cl=1,rt=4,cr=1,rb=4,text="- Phones")
c6=(cl=2,rt=4,cr=2,rb=4,text="11.2",type=NUMERIC)
```

2.2. Logical Layer

`Entry` object serves to represent a data value of a table. It consists of the following attributes: `value` — a value (text), `labels` — a set of labels associated with this entry, and `cell` — the physical layer reference to a cell as its origin that serves as data provenance. An entry can be associated with only one label in each category.

`Label` represents a label (key) that addresses one or more entries (data values). It is defined as follows: `value` — a value (text), `children` — a set of labels which are children of this label, `parent` — its parent label, `category` —

an associated category, `cell` — the physical layer reference to a cell as its origin (data provenance).

`Category` models a category of labels as follows: `name` — an internal name, `uri` — a uniform resource identifier representing this category (concept) in an external vocabulary, `labels` — a set of its labels. Each label is associated with only one category. Labels combined into a category can be organized as one or more trees.

This layer allows representing items differently depending on target requirements of the table transformation. For example, one of the possible target can generate the entry (`e1`), labels (`l1`,...,`l5`), and categories (`d1`,...,`d4`) for the table shown in Fig. 1, *a* as follows:

```
e1=(value="11200", labels={l1,l2,l3,l5}, cell=c6)
l1=(value="retail", category=d1, cell=c2)
l2=(value="2016", category=d2, cell=c3)
l3=(value="u.s. dollars", category=d3, cell=c3)
l4=(value="electronics", children={l5,...}, category=d4, cell=c4)
l5=(value="phones", parent=l4, category=d4, cell=c5)
d1=(name="SALE CHANNEL", labels={l1,...})
d2=(name="FISCAL YEAR", labels={l2,...})
d3=(name="CURRENCY", labels={l3,...})
d4=(name="PRODUCT", labels={l4,l5,...})
```

They can be presented as a tuple of a target relational table (Fig. 1, *b*).

3. Table Analysis and Interpretation Rules

The rules expressed in our language are intended to map explicit features (layout, style, and text of cells) of an arbitrary table into its implicit semantics (entries, labels, and categories) Fig. 3 demonstrates its grammar in Extended Backus-Naur form. A rule begins with the keyword `rule` and ends with `end`. A number that follows the keyword `rule` determines the order of executing this

```
rule       = 'rule' <a Java integer literal>
             'when' condition 'then' action 'end' <EOL> {rule} <EOF>
condition  = query identifier [':' constraint {',' constraint}
             [',' assignment {',' assignment}]] <EOL> {condition}
constraint = <a Java boolean expression>
assignment = identifier ':' <a valid Java expression>
query      = 'cell' | 'entry' | 'label' | 'category' |
             'no cells' | 'no entries' | 'no labels' | 'no categories'
action     = merge | split | set text | set indent | set mark |
             new entry | new label | add label | set parent |
             set category <EOL> {action}
merge      = 'merge' identifier 'with' identifier
split      = 'split' identifier
set text   = 'set text' <a Java string expression> 'to' identifier
set indent = 'set indent' <a Java integer expression> 'to' identifier
set mark   = 'set mark' <a Java string expression> 'to' identifier
new entry  = 'new entry' identifier ['as' <a Java string expression>]
new label  = 'new label' identifier ['as' <a Java string expression>]
add label  = 'add label' identifier | (<a Java string expression> 'of'
             identifier | <a Java string expression>) 'to' identifier
set parent   = 'set parent' identifier 'to' identifier
set category = 'set category' identifier | <a Java string expression>
               'to' identifier
identifier   = <a Java identifier>
```

Figure 3: Grammar of CRL in Extended Backus-Naur form.

rule. The left hand side (`when`) of a rule consists of one or more *conditions* that enable to query available facts which are cells, entries, labels, and categories of a table. Each of the conditions listed in the left hand side of a rule has to be true to execute its right hand side (`then`) that contains *actions* to modify the existed or to generate new facts about the table.

3.1. Conditions

We use two kinds of conditions. The first requires that there exists at least one fact of a specified data type, which satisfies a set of constraints:

```
cell variable: constraints, assignments
entry variable: constraints, assignments
label variable: constraints, assignments
category variable: constraints, assignments
```

The condition 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 `variable`, a variable of the specified fact type. The third optional part begins with the colon character. It defines constraints for restricting the requested facts and assignments for binding additional variables with values. A *constraint* is a boolean expression in Java. The comma character separating the constraints is the logical conjunction of them. An *assignment* (`variable: value`) sets a value (Java expression) to a variable. A condition without constraints allows querying all facts of specified type.

The second kind of conditions determines that there exist no facts of a specified type, which satisfy a set of constraints:

```
no cells: constraints
no entries: constraints
no labels: constraints
no categories: constraints
```

The firt part of these conditions is a keyword for satisfying a type of facts. The second part contains constraints on the facts.

3.2. Cell Cleansing

In practice, hand-coded tables often have messy layout (e. g. improperly splitted or merged cells) and content (e. g. typos, homoglyphs, or errors in indents). We address several actions to the issues of cell cleansing, that can be used as the preprocessing stage.

3.2.1. Cell merging

Two cells can be merged when they share one border. The action combines two adjacent cells `$cell1` and `$cell2` into the one merged cell:

```
merge $cell1 with $cell2
```

As a result, the addressee `$cell2` becomes a merged cell with new coordinates that span both cells.

3.2.2. Cell splitting

This action allows dividing a merged cell `$cell` that spans n-tiles into n-cells.

```
split $cell
```

Each of the n-cells completely copies content and style from the merged cell and coordinates from the corresponding tile.

Example 1. The table shown in Fig. 4, *a* contains the merged cells ('1', '4', and '5'). We can split them, using the following rule:

```
when
  cell $cc: cl == 1, rt == 1, blank
  cell $c: cl > $cc.cr, rt > $cc.rb
then
  split $c
```

As a result, the table (Fig. 4, *a*) is transformed into the table (Fig. 4, *b*).

3.2.3. Cell content modification

There are two actions modifying cell content. The first sets a new value `string_value` to a cell `$cell`:

```
set text string_value to $cell
```

Some string processing (e. g. regular expressions and string matching algorithms) implemented as Java-methods can be involved in the action.

The second one modifies the indent value `integer_value` of a cell `$cell`:

```
set indent integer_value to $cell
```

3.3. Role Analysis

This stage aims to recover entries and labels as functional data items presented in tables. We also enable associating cells with user-defined marks (tags) that can assist in both role and structural analysis.

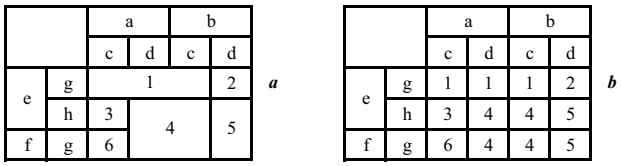


Figure 4: Pivot tables with empty stub heads. We depict here entries as numbers and labels as Latin characters.

3.3.1. Cell marking

The action provides marking a cell `$cell` with a word or phrase `string_value`:

```
set mark string_value to $cell
```

The assigned mark can substitute the corresponding constraints in subsequent rules. The typical practice is to set a mark to all cells, which play the same role or are located in the same table functional region. Thereafter, we can use these marks in subsequent rules instead of repeating constraints on cell location in the regions.

Example 2. Looking at the pivot tables shown in Fig. 4, we can assume that each of them has an empty cell (stub head region) located in the top-left corner. It can be considered as a "critical cell" (Nagy, 2012) which determines three functional regions: body, head, and stub.

The rule based on the assumption adds the mark (`body`) to each cell `$c` located in the body.

```
when
  cell $cc: cl == 1, rt == 1, blank
  cell $c: cl > $cc.cr, rt > $cc.rb
then
  set mark "body" to $c
```

Similarly, we can write rules for marking the corresponding cells with the words: `head` and `stub`. As a result of matching these rules against the cells of the table (Fig. 4, *b*), we recover the following facts:

```
 c1=(cl=3, rt=3, cr=3, rb=3, value="1", mark="body"),...,
c12=(cl=6, rt=5, cr=6, rb=5, value="5", mark="body"),
c13=(cl=3, rt=1, cr=4, rb=1, value="a", mark="head"),...,
c18=(cl=6, rt=2, cr=6, rb=2, value="d", mark="head"),
c19=(cl=1, rt=3, cr=1, rb=4, value="e", mark="stub"),...,
c23=(cl=2, rt=5, cr=2, rb=5, value="g", mark="stub")
```

3.3.2. Entry and label generating

Two actions presented below generate entries and labels in a cell `$cell`, using string expressions `entry_value` and `label_value` usually obtained as a result of string processing its textual content:

```
new entry $cell as entry_value
new label $cell as label_value
```

The following short form creates an entry and a label from the cell text:

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

Example 3. The bilingual table (Fig. 5, *a*) duplicates labels in two languages (Greek and Latin symbols). Assuming that the first label (word) in a cell is written in one language and the second in other, we can use the rule below to generate two labels from each cell located in the leftmost column or the topmost row:

```
when
  cell $c: cl==1 || rt==1, !blank
then
  new label $c as token($c, 0)
  new label $c as token($c, 1)
```

In this example, we expect that the function `token` is implemented as a Java-method and imported into the rules. It returns a token (word) specified by an index from text of a cell.

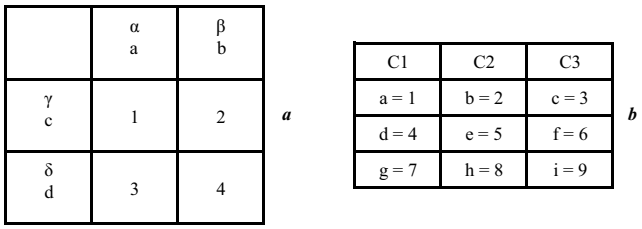


Figure 5: Tables with cells containing more than one entry or label: bilingual table, where each non-empty cell has either two labels or two entries (a); text like as "key=value" in a cell can be interpreted as a label-entry pair, where key-part is a label and value-part is an entry (b).

For the table (Fig. 5, a) this rule generates 8 labels:

```
l1=(value="α"), l2=(value="a"),..., l7=(value="δ"), l8=(value="d")
```

For tables similar to the one shown in Fig. 5, 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 of the text:

```
when
  cell $c: rt > 1
then
  new label $c as left($c, '=')
  new entry $c as right($c, '=')
```

The functions `left` and `right` are implemented as Java-methods. In the presented case, they extract substrings before and after the character (`"="`) respectively.

For the table (Fig. 5, b) this rule generates 9 entries and 9 labels:

```
e1=(value="1"),..., e9=(value="9"),
l1=(value="a"),..., l9=(value="i")
```

3.4. Structural Analysis

The next stage recovers pairs of two kinds: entry-label and label-label.

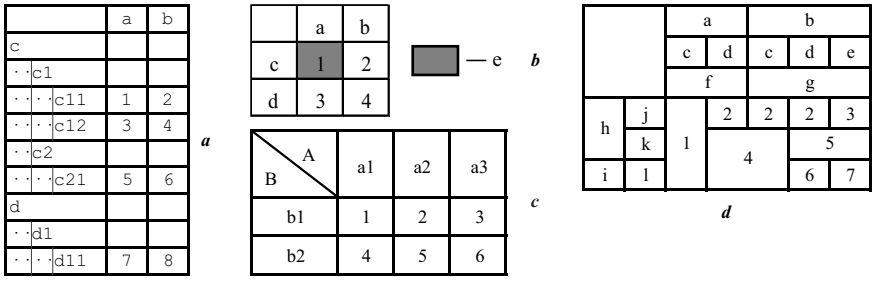


Figure 6: The table with indentation in the stub for indicating a row label hierarchy (a); the table where the gray fill color of the cell is used as a reference to the footnote 'e' (b); the table containing two category names: 'A' and 'B' (c); the table with grouped labels (d).

3.4.1. Entry-label associating

The action below binds an entry `$entry` with a label `$label`:

```
add label $label to $entry
```

There are two additional ways to create an entry-label pair. The first associates an entry `$entry` with a label specified by its value (`label_value`) from a category indicated by its name (`category_name`):

```
add label label_value of category_name to $entry
```

The second creates a pair between them similarly but using a defined category `$category`:

```
add label label_value of $category to $entry
```

In both cases, we try to find or create the label in the specified category.

Example 4. The table in Fig. 6, b depicts the use of a cell background color (`Color.GRAY`[3] in the cell `B2`) as a reference to the footnote ("e"). We can recover this relationship, using the style features as follows:

```
when
  entry $e: cell.style.bgColor == Color.GRAY
```

[3] `https://docs.oracle.com/javase/7/docs/api/java/awt/Color.html`

```
then
  add label "e" of "footnotes" to $e
```

We assume that the following facts exist before executing the rule:

```
c1=(style.bgColor=Color.GRAY, entries={e1}),
e1=(value="1", cell=c1)
```

As a result, we generate the new facts after its execution:

```
e1=(value="1", cell=c1, labels={l1}), l1=(value="e", category=d1),
d1=(name="footnotes", labels={l1})
```

3.4.2. Label-label associating

This action connects two labels `$label1` as a parent and `$label2` as its child:

```
set parent $label1 to $label2
```

Example 5. A header located in the stub (leftmost column) often begins with an indent presented as a series of spaces, dots, or other padding characters. Usually, the indents denote hierarchical label-label pairs. For example, when each level in a label hierarchy augments the indents with two additional dots (Fig. 6 *a*), we can recover label-label pairs as follows:

```
when
  cell $c1: cl == 1
  cell $c2: cl == 1, rt > $c1.rt, indent == $c1.indent + 2
  no cells: cl == 1, rt > $c1.rt, rt < $c2.rt, indent == $c1.indent
then
  set parent $c1.label to $c2.label
```

As a result, we recover the following label-label pairs:

```
(c,c1), (c1,c11), (c1,c12), (c,c2), (c2,c21), (d,d1), (d1,d11)
```

3.5. Interpretation

The stage includes actions for recovering label-category pairs.

3.5.1. Label categorizing

The action of label categorizing consists in associating a label `$label` with a category `$category`:

```
set category $category to $label
```

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

```
set category category_name to $label
```

In the latter case, we try to find or create the category with this name.

Example 6. Some tables contain category names among their headings. The stub head of the table shown in Fig. 6, *c* contains two category names: 'A' is for the category of the column labels ('a1', 'a2', and 'a3') and 'B' is for the category of the row labels ('b1' and 'b2'). The names can be used to create corresponding categories and to categorize labels. In case of tables similar to the one shown in Fig. 6, *a*, we can assume that a cell in the top-left corner (stub head) contains two category names: the first one describes column labels and the second one addresses row labels.

The rule below creates a category from the first token (word) contained in the top-left corner cell and uses it to categorize column labels:

```
when
  cell $cc: cl == 1, rt == 1
  label $l: cell.rt == 1
then
  set category token($cc, 0) to $l
```

In case of the table (Fig. 6, *c*), we generate the following facts:

```
l1=(value="a1", category=d1), l2=(value="a2", category=d1),
l3=(value="a3", category=d1), d1=(name="A", labels={l1, l2, l3})
```

3.5.2. Label grouping

Arbitrary tables often place all labels of one category in the same row or column. Consequently, we can suppose that the labels belong to a category without defining its name. In the cases, grouping two or more labels means that they all belong to an undefined category. The action places two labels `$label1` and `$label2` in one group:

```
group $label1 with $label2
```

All labels of a group can be associated with only one category.

Example 7. The stub of the pivot table (Fig. 6, *d*) consists of two columns. We can suppose that all stub labels originated from one column belong to the same undefined category. The rule below arranges its stub labels into two groups (`{h, i}` and `{j, k, l}`):

```
when
  label $l1: cell.mark == "stub"
  label $l2: cell.mark == "stub", cell.rt == $l1.cell.rt
then
  group $l1 with $l2
```

4. Implementation

We develop TabbyXL[4], a command-line tool for spreadsheet data canonicalization that implements our methodology. Its architecture is illustrated in Fig. 7. The data structures for representing table facts (cells, entries, labels, and categories) are Java classes developed in accordance with the naming conventions of JavaBeans[5] specification. Table facts are instances of these classes. This allows using any rule engine implemented JSR 94: Java Rule Engine API[6] specification.

[4] https://github.com/cellsrg/tabbyxl

[5] http://www.oracle.com/technetwork/java/javase/tech/spec-136004.html

[6] http://jcp.org/en/jsr/detail?id=94

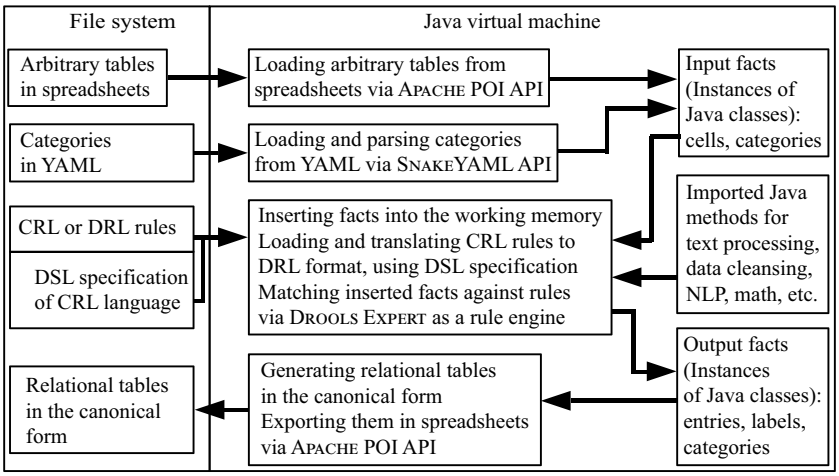


Figure 7: Architecture of TabbyXL, the tool for spreadsheet data canonicalization.

We use Drools Expert[7] rule engine. Table analysis and interpretation rules can be expressed in DRL, the native format of Drools Expert, or CRL, our domain-specific language. CRL rules are preliminarily translated into DRL, using DSL specification with mappings from CRL to DRL constructs. Moreover, CRL rules can be enriched by DRL rule attributes (e. g. for rule activation `lock-on-active` or `no-loop true`) and commands for updating the working memory (DRL method `update`).

The process of table canonicalization begins with loading tabular data from Excel spreadsheets via Apache POI API[8]. Our tool requires that the pair of tags `$START` and `$END` point out to location of each input table. It allows avoiding detecting tables in a spreadsheet. Each table forms a set of initial cells. They are asserted as facts in the working memory of the rule engine. The input can also be enriched by categories as facts specified in YAML[9] format. Each input category consists of its name, a set of labels, and a set of constraints

[7] `http://drools.jboss.org/drools-expert`
[8] `http://poi.apache.org`
[9] `http://yaml.org`

defining ranges of permissible label values (Shigarov et al., 2016b).

The rule engine matches available facts against loaded rules. As a result, we generate new facts. Recovered semantics (entries, labels, and categories) allows transforming source tabular data into the canonical form. The process ends with that we export each canonical table to spreadsheet format.

The canonical form requires that the topmost row contains field (attribute) names. Each of the remaining rows is a record (tuple). It obligatorily includes the field named `DATA` that contains entries. Each extracted category constitutes a field that contains its labels. Each record presents recovered relationships between an entry and labels in each category.

4.1. User Study

We conducted a user study to evaluate how well potential users can develop CRL rules. We invited 10 practitioners in software engineering, data management, and data analytics to participate in our user study. Each of them had a background of programming in at least one general-purpose language. Only two participants used rule-based languages in their practice.

First, we introduced a tutorial that briefly explained our table object model and demonstrated simple CRL rules for the table understanding stages. Second, we asked each participant to develop a rule-set for transforming source tables of one type (Fig. 8, *a*, *c*) into the target ones (Fig. 8, *b*, *d*). The presented source tables were used in a real-world application of collecting information on electrical and technical equipment in a power company.

We required that participants implemented only 7 rules for the following tasks: (1) data cleansing, (2) entry generation, (3) label generation, (4) associating entries with column labels, (5) associating entries with row labels, (6) categorizing column labels, and (7) categorizing row labels. The reference rule-set is shown in Fig. 9.

Each participant developed a complete rule-set using a simple text editor. Only 3 participants made 8 syntax errors, and 6 made 14 semantic errors. In several cases, they also used syntactically and semantically correct but redun-

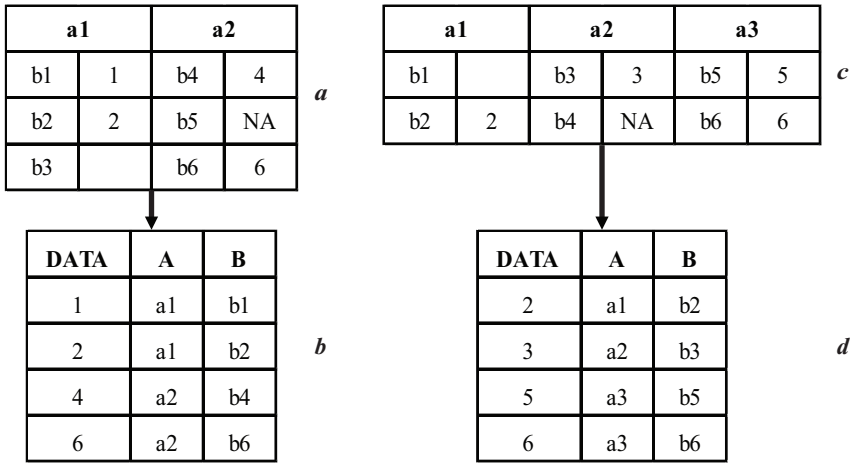


Figure 8: Source tables (a, c) and their target canonical forms (b, d) used in the user study, that satisfy the following assumptions: $1, \ldots, n$ are entries, $a_1, \ldots, a_m$ are column labels of the category A, $b_1, \ldots, b_k$ are row labels of the category B.

(1)
```
when cell $c: text == "NA"
then set text "" to $c
```

(2)
```
when cell $c: (cl % 2) == 0, !blank
then new entry $c
```

(3)
```
when cell $c: (cl % 2) == 1
then new label $c
```

(4)
```
when
  entry $e
  label $l: cell.cr == $e.cell.cr
then add label $l to $e
```

(5)
```
when
  entry $e
  label $l: cell.rt == $e.cell.rt, cell.cl == $e.cell.cl - 1
then add label $l to $e
```

(6)
```
when label $l: cell.rt == 1
then set category "A" to $l
```

(7)
```
when label $l: cell.rt > 1
then set category "B" to $l
```

Figure 9: A reference rule-set for transforming the source tables (Fig. 8, a, c) to the target canonical forms (Fig. 8, b, d).

dant constraints. In spite of only 3 users developed the rules without errors, the rest participants were able to correct the errors in process of the rule-set compilation and execution.

We believe that our tool is useful not only for computer scientists in the area of table understanding but also for practitioners in data management and analytics. The user study shows that qualified users such as database administrators or data analysts are able to design and implement programs (rule-sets) for specific tasks of spreadsheet data transformation.

5. Case Study

The purpose of the experiment is to show a possibility of using our tool[10] for tables, which originate from various sources produced by different authors but pertain to the same document genre. The experiment includes two parts: (i) designing and implementing an experimental rule-set for tables of the same genre, and (ii) evaluating the performance of the rule-set on a set of these tables. All data and rules for reproducing the experiment results are available as the published dataset[11] (Shigarov, 2017).

5.1. Dataset

TROY200 (Nagy, 2016), the existing dataset of tables, satisfies the purpose of our experiment. It contains 200 arbitrary tables as CSV (Comma-Separated Values) files collected from 10 different sources of the same genre, government statistical websites predominantly presented in English language. We use its earlier version[12] that stores these tables with style features (fonts, alignment, and indentation) as spreadsheets.

Each table of the dataset contains four cell regions, which correspond to three roles: category names, labels, and entries (Fig. 10). A region can have one of the appropriate layouts shown in Fig. 10. Table 1 presents the distribution of these tables by the layouts.

5.2. Source and Target

The role and structural analysis depend on both features of source tables and objectives of table processing. Our examination reveals that most tables presented in the dataset meet the following *source requirements*:

1. A table consists of four cell regions having different functions and separated by two invisible perpendicular lines as shown in Fig. 10, *a*: (i) each

[10] TabbyXL v0.1, `https://github.com/cellsrg/tabbyxl/releases`
[11] `https://data.mendeley.com/datasets/448jdx7gcr/1`
[12] `http://tango.byu.edu/data`

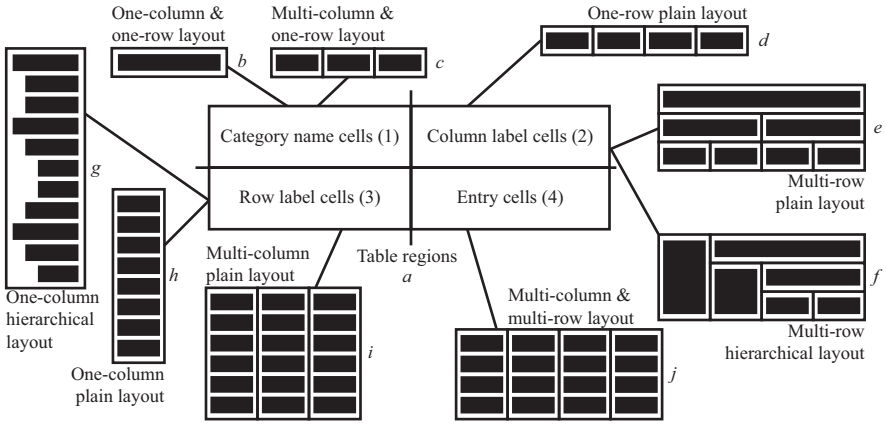


Figure 10: Cell regions (a) and their layouts (b–j) in tables of the experiment dataset.

non-empty cell of the top-left region contains a category name; (ii) each non-empty cell of the top-right region contains a column label; (iii) each non-empty cell of the bottom-left region contains a row label; (iv) each non-empty cell of the bottom-right region contains an entry.

2. Each cell region has one of the appropriate layouts as shown in Fig. 10 and enumerated in Table 1.
3. Head cells can form a hierarchy of labels when their columns span and nest (Fig. 10, e and f).
4. Row labels located in the leftmost column can form a hierarchy (Fig. 10, g). Three typographical ways can denote the presence of a label hierarchy: (i) each level of label nesting appends one additional indent equaled two spaces; (ii) hyphen char ('-') at the beginning of a label indicates that the label is nested; (iii) text highlighted by the bold font can signalize spanning label.
5. Data cells contain either numeric values or special words (e. g. '#', 'x').

Our aim consists in transforming the tablular data to a general-purpose target representation that can go through a further interpretation. It provides the following *target requirements*:

1. All labels originated from head cells are placed into one hierarchical cat-

Table 1: Using cell region layouts in tables of the experiment dataset

Region	Layout	Fig. 10	Cases
(1) Category name cells	One-column one-row	*b*	94.5%
	Multi-column one-row	*c*	5.5%
(2) Column label cells	One-row plain	*d*	65.5%
	Multi-row plain	*e*	26%
	Multi-row hierarchical	*f*	8.5%
(3) Row label cells	One-column hierarchical	*g*	47.5%
	One-column plain	*h*	47%
	Multi-column plain	*i*	5.5%
(4) Entry cells	Multi-column multi-row	*j*	100%

egory (`CalCat` fields in Fig. 11, *b* and *d*).

2. All labels obtained from the same column of a stub are placed into a separate hierarchical category (`RowCat` fields in Fig. 11, *b* and *d*).
3. Footnotes and superscript text are ignored.

5.3. Rule-Set

We have designed and implemented a rule-set that transforms the tables of the dataset into the canonical form according to the presented target requirements. It includes 16 rules that can be grouped as follows:

1. *Preprocessing.* `Rule-1` replaces each cell value matching "#" or "s" on "0". `Rule-2` replaces each cell value such as "F", "x", "NA", horizontal ellipsis, or dash on `null`. `Rule-3` replaces dashes or hyphens at the beginning of a negative number on the regular minus symbol.
2. *Generating and associating column labels.* We use two rules to generate and associate labels in a head. The first `Rule-4` generates labels from non-empty cells of the topmost row. The second `Rule-5` searches for head rows top down beginning from the topmost one, examining pairs of neighbor rows. It expects that column labels in the pair form a hierarchy (Fig. 10, *e*

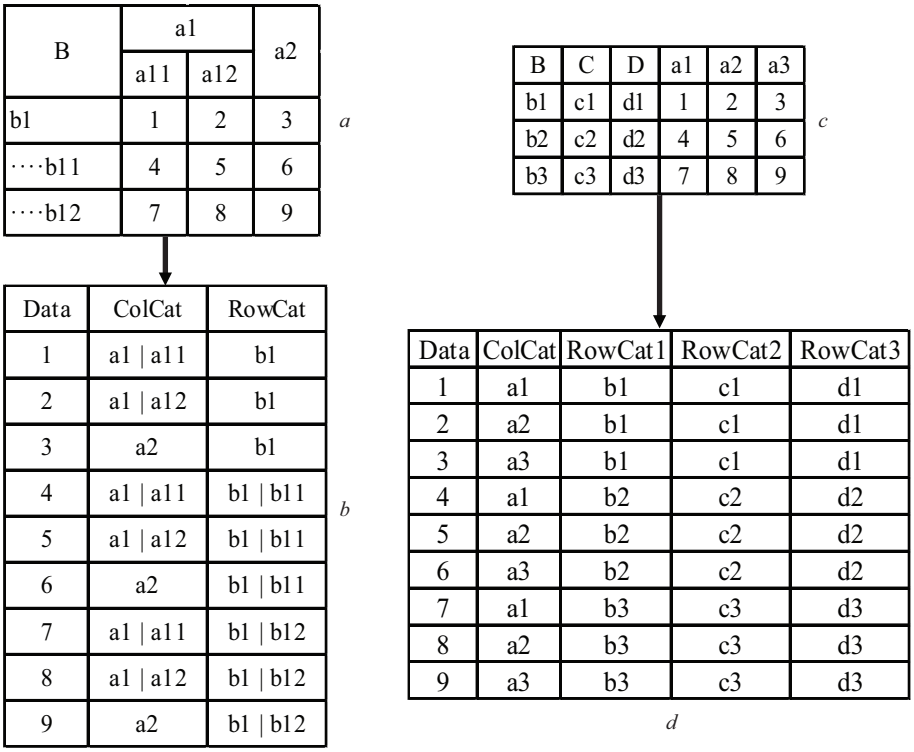


Figure 11: Two arbitrary tables (a and c) and their canonical forms that satisfy the experimental target requirements (b and d).

and f). If a non-empty cell c located on i-row in the top-right region spans several columns and non-empty cells $c_1,\ldots,c_n$ are located in these nested columns on $i+1$-row, then the cell c contains a parent label for labels produced from the cells $c_1,\ldots,c_n$.

3. *Generating row labels.* `Rule-6` creates labels from non-empty cells in the leftmost column. `Rule-7` continues to generate labels from cells located below the head in case when their values are not numbers.
4. *Generating entries.* `Rule-8` generates entries from numeric values of the remaining cells located below the head.
5. *Associating row labels.* First, we use `Rule-9` to fix indentation. It sets up the indent for each cell when its string value begins from a hyphen-minus

character. The next two rules recover a hierarchy of row labels located in the leftmost column (Fig. 10, g). `Rule-10` searches for parent-child pairs among row labels assuming that each new level of a hierarchy appends one additional indent (two spaces) in cells of nested labels. `Rule-11` associates a child label with parent one when the parent cell with a boldface text is located above the child cell with a regular text. Exceptions are made for third cases where a parent candidate is "Total", "All", or "I alt". They are ignored.

6. *Categorizing labels.* The three rules (`Rule-12`, `Rule-13` and `Rule-14`) place labels into synthetic categories: `ColCat`, `RowCat1`,..., `RowCatN`.
7. *Associating entries.* `Rule-15` associates an entry with a terminal label when they are originated from cells located in the same column. `Rule-16` connects an entry and all labels when their cells are in the same row.

5.4. Performance Evaluation

The case study includes the ground-truth data that we have collected from Troy200 tables. The performance evaluation is based on comparing the ground-truth data with the tables generated by executing the presented rule-set for the experiment dataset.

5.4.1. Ground-Truth Data

The collected ground-truth data covers both role and structural analysis. Their form is designed for human readability. Moreover, they are independent of the presence of critical cells in tables.

Each table from the dataset was accompanied by two recordsets: `ENTRIES` and `LABELS`. The first specifies entries. Each record is a triple: an entry, its provenance (cell address in its source table), and a set of associated labels. For example, a fragment of `ENTRIES` recordset is shown below:

```
ENTRY  PROVENANCE  LABELS
243    T11         "2002 [B11]", "balance [T4]"
2871   S11         "2002 [B11]", "imports [S4]"
```

Table 2: Experiment results on the role analysis stage.

	type of instances		
	entries	labels	both
Number of			
correct instances	16602	4849	21451
existing instances	16918	4859	21777
recovered instances	16602	5178	21780
Recall	0.9813	0.9979	0.9850
Precision	0.1000	0.9365	0.9849

Each record in the second recordset presents a label, including a reference to its parent. We demonstrate an example of `LABELS` recordset below:

```
LABEL    PROVENANCE  PARENT
balance  T4          other ict goods [R3]
imports  S4          other ict goods [R3]
```

5.4.2. Measures

To evaluate our rule-set we adapt the well-known measures: *recall* and *precision*. When R is a set of instances in the result table and S is a set of instances in the corresponding source table, then:

$$\text{recall} = \frac{|R \cap S|}{|S|} \quad \text{precision} = \frac{|R \cap S|}{|R|}$$

An instance refers to an entry, label, entry-label pair, or label-label pair. So, these measures can be separately calculated for each type of instances (entries, labels, entry-label pairs, or label-label pairs) as well as for combinations of these types.

5.4.3. Experiment Results

To evaluate the role analysis we calculate the recall and precision for entries, labels, and the sum of these both types (Table 2). The performance evaluation at the structural analysis stage is determined as measuring the recall and precision

Table 3: Experiment results on the structural analysis stage.

	type of instances		
	entry-label pairs	label-label pairs	both
Number of			
correct instances	34270	1951	36221
existing instances	35066	2078	37144
recovered instances	34386	1994	36380
Recall	0.9773	0.9389	0.9751
Precision	0.9966	0.9784	0.9956

for entry-label pairs, label-label pairs, and pairs of both types (Table 3). The presented results have been obtained automatically through our command-line application included in TabbyXL as `Evaluator.class`[13].

5.4.4. Errors

Among 200 tables of the dataset, only 25 are processed with errors (1249 false negatives in 25 tables, and 488 false positives in 14 ones). Notice that, one table[14] is not processed. It results in 948 false negative and 316 false positive errors, which amount to about 73% of all errors. Its entries are not recovered because they are not numeric as it is assumed in the used rule-set.

For the rest of cases, the errors came from a stub or head. Their examples are illustrated in Fig. 12. We classify them according to their causes. There are five types of the causes originated from stub (Fig. 12 *a*).

1. *Cut-in.* A header in a row, where all rest cells are empty, can be a root for other headers or just a leaf.
2. *Multi-cell header.* Text parts of a header can be placed into several neighbor cells.

[13] `https://github.com/cellsrg/tabbyxl/blob/master/src/main/java/ru/icc/cells/ssdc/evaluation/Evaluator.java`

[14] file C10082.csv in Troy200 dataset

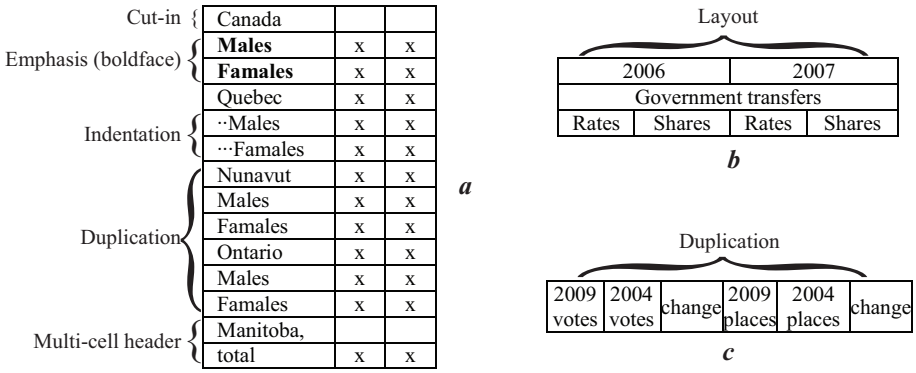


Figure 12: Causes of errors in stubs (a) and heads (b) and (c).

3. *Indentation.* A header hierarchy can be constituted by untidy indents (e. g. padding characters differently aligned).
4. *Emphasis.* The boldface font can be used to emphasize a root in a header hierarchy or data aggregation.
5. *Repeating.* A header hierarchy has one or more sets of repeating headers without visual emphasizing its levels.

The errors from a head are produced for the following causes (Fig. 12 b and c).

1. *Layout.* A layout of cells is not expected in our rule-set (e. g. a spanning header located under nested ones).
2. *Duplication.* There are two or more headers with the same text, i. e. additional headers are needed to read them.

Table 4 shows the distribution of the errors that arise in subs or heads. Among 24 tables processed with errors only one has two types of error causes ("indentation" and "multi-cell header"). Each of the remaining tables is accompanied by only one of the enumerated types.

The presented results show that stubs and heads are complex objects for an automatic analysis and interpretation. On the one hand, the same layout or style features of two stubs or two heads can have different meaning (e. g. the emphasis

Table 4: Causes of errors originated from stubs or heads.

Origin	Amount of tables	False negatives	False positives
Stub	21	153	84
Cut-in	10	34	3
Multi-cell header	4	48	41
Indentation	4	30	14
Emphasis (boldface)	2	30	26
Repeating	2	11	0
Head	3	148	88
Layout	2	130	70
Duplication	1	18	18
Total	24	301	172

referencing to a header hierarchy or data aggregation). On the other hand, the absence of the features also leads to errors in the table understanding. Perhaps, these objects require rigorous studying and developing specialized algorithms for their analysis.

6. Comparison with Others

We experimentally compare our tool with two state of the art solutions for spreadsheet table understanding:

- MIPS (Minimum Indexing Point Search), an algorithm for table segmentation into the functional cell regions, proposed by Embley et al. (2016) as a part of their algorithmic solution for end-to-end transforming "header-indexed" tables converted in CSV format to a relational form ("category tables").

- Senbazuru, a spreadsheet database management system, proposed by (Chen & Cafarella, 2014; Chen, 2016) for extracting relational data from spreadsheets ("data frames").

Their goals are similar to ours. Both solutions aim to transform spreadsheet tables with a complex structure into a relational form. They also involve the stages of role and structural analysis. Both rely on similar functional cell regions. Embley et al. (2016) define four regions `StubHeader` (stub head), `ColHeader` (head), `RowHeader` (stub), and `Data` (body). Chen & Cafarella (2014) focus on data frames, a common data model in spreadsheets. A data frame consists of three regions: *top annotation* (head), *left annotation* (stub) and *data* (body). Both systems also support hierarchical relationships of headings (labels).

The considered systems limit a range of tables, which can be processed successfully, by the assumptions on the functional cell regions. In contrast to these systems, we clearly divide physical and logical table structures on the object level (Section 2). Our model associates roles with data items (entries and labels) instead of cells or regions (e. g. head, stub, or body). Therefore, our tool does not have the limitations of the comparable systems.

We demonstrate the experimental results that are close to ones of the state of the art systems on the existing datasets of real-world tables. Unlike these solutions, our tool allows developing and executing rule-sets to process others types of tables, which do not fit in the presented models ("header-indexed tables" or "data frames").

6.1. Role Analysis

We compare our tool with MIPS in role analysis. MIPS exploits the assumption that there exists MIP (Minimum Indexing Point) in a table that determines boundaries of its regions: stub, head, and body. They successfully identify roles of the cell region when MIPS finds MIP correctly.

Embley et al. (2016) evaluate only table segmentation into the functional cell regions on Troy200 dataset (Nagy, 2016) (briefly described in Section 5.1). They report that one of the 200 Troy200 tables is trivial (it contains only one data column). Among the 199 non-trivial tables, MIP is correctly detected in 197 cases. Therefore, the *accuracy* of automatic table segmentation (i. e. detection of a stub, head, and body) by MIPS reaches 0.9899 (197/199) on

TROY200 dataset.

We evaluate the rule-set presented in Section 5.3. It enables transforming TROY200 tables into the canonical form. We also annotate not blank cells by tags `ColumnHeading`, `RowHeading`, and `DataCell`, which can be considered as tags for cells of a head, stub, and body respectively. Therefore, we identify which of cells belong to each of these functional regions.

In our experiment, total 7 tables are processed with errors of role analysis. However, only 2 of them are processed with errors, when our rule-set does not detect a head, stub, and body successfully. The *accuracy* of our table segmentation is 0.9950 (198/199) on TROY200 dataset without one trivial table noted by Embley et al. (2016). It amounts 0.9900 (198/200) for all tables.

The presented metrics conditionally show the correctness of role analysis. However, some errors can occur in cases when tables are successfully segmented (Section 5.4.4). We propose to evaluate recall and precision of recovering entries and labels. We demonstrate the following results of recovering entries and labels on TROY200 dataset (Section 5.4.3): *recall* = 0.9850, *precision* = 0.9849, and $F_1 = 0.9849$.

6.2. Structural Analysis

We compare our tool with the automatic extractor of SENBAZURU in structural analysis. SENBAZURU extracts parent-child pairs of labels located in a head (top annotation region) and stub (left annotation region) based on an undirected graphical model. It exploits graphical style features (e. g. font, alignment, and indentation) as well as correlated extraction decisions.

Chen & Cafarella (2014) evaluated accuracy of the automatic extraction in predicting correct parent-child relationships by using standard metrics of *recall*, *precision*, and F_1. Their experiment is based on two spreadsheet corpora[15]. One of them is SAUS (The 2010 Statistical Abstract of the United States) that consists of 1369 spreadsheets. They randomly selected 200 tables from SAUS

[15] `http://web.eecs.umich.edu/~michjc/structuredweb/index.html`

and then split them into two equal parts for training and testing the automatic extractor. They reached the following results: *recall* = 0.8860, *precision* = 0.8860, and $F_1 = 0.8860$.

Our experiment exploits a subset of 200 SAUS tables randomly selected and listed by Nagy (2016). We randomly split this subset into two equal parts. The first part was used to develop a rule-set for processing SAUS tables. This rule-set includes 14 rules, 7 of which recover parent-child pairs of labels. We evaluate the rule-set only in recovering parent-child pairs using the second part of the subset.

The process of evaluation is implemented as follows. Two experts independently compare source tables and their automatically generated canonical forms. They examine that each parent-child pair is processed successfully or not. When they make opposite decisions on a pair, then a final decision is a consensus between them. We have obtained the following results: *recall* = 0.7879 (3217/4083), *precision* = 0.9606 (3217/3349), and $F_1 = 0.8657$.

Our result (F_1) is less only by 0.0233 than one shown by the specialized system SENBAZURU. The many errors caused by inaccurate table markup (physical cell structure) reduce the recall of our tool. There are two main failure reasons in our experiment: (i) in table heads, one human-readable (visual) cell is placed in several physical cells, (ii) in table stubs, the content of one label is distributed in several consecutive cells.

We believe that physical cell structure cleansing (repair) as a preprocessing stage will allow avoiding many of the errors and significantly improve the recall in recovering label-label pairs for tables similar to SAUS ones. We obtain *recall* = 0.9389, *precision* = 0.9784, and $F_1 = 0.9582$ for recovering parent-child (label-label) pairs on TROY200 dataset (Section 5.4.3), where tables have more accurately markup, i. e. the human-readable (visual) head structure conforms physical one.

7. Related work

Many methods and tools, where table analysis and interpretation serve to extract and transform tabular data to a machine-interpretable representation, e. g. relational databases, RDF[16], or OWL[17], were suggested in recent years.

7.1. Table analysis and interpretation methods

The methods for table analysis are proposed in the papers (Pivk et al., 2007; Abraham & Erwig, 2007; Kim & Lee, 2008; Tao & Embley, 2009; Seth & Nagy, 2013; Mauro et al., 2013; Adelfio & Samet, 2013; Chen & Cafarella, 2013; Chen et al., 2013; Chen & Cafarella, 2014; Embley et al., 2014; Nagy et al., 2014; Embley et al., 2016; Rastan et al., 2016; Goto et al., 2016; Koci et al., 2016). Each of them is designed for a few widespread types or features of arbitrary tables.

Pivk et al. (2007) involve heuristics on structure and textual content of a table, which are designed for three typical table types. Abraham & Erwig (2007) combine different heuristic-based algorithms that classify spreadsheet cells into four functional groups (roles). Kim & Lee (2008) use the analysis of spatial, style, and textual information from web tables based on embedded rules and regular expressions for five table types. Tao & Embley (2009) assume that all content in one cell is either a label or an entry. Several works (Mauro et al., 2013; Nagy et al., 2015; Embley et al., 2016; Nagy & Seth, 2016) use the critical cells that allow dividing a table into four functional regions (head, stub, stubhead, and body) (Nagy, 2012). Adelfio & Samet (2013) exploit common table patterns based on sequences of row labels which correspond to roles (header, data, aggregate, etc). The papers (Chen & Cafarella, 2013, 2014; Rastan et al., 2016; Goto et al., 2016) focus on the structural analysis for hierarchical headers in the stub. Koci et al. (2016) propose the classification approach to identify five

[16] `https://www.w3.org/RDF`
[17] `https://www.w3.org/OWL`

typical functional blocks of tables, using a wide range of cell features presented in spreadsheets.

The methods listed above are based on various assumptions about layout, style and textual content of widespread tables. Typically, they implement the assumptions as rules or heuristics incorporated in their algorithms. We think that there are many specific features beyond the types covered by these methods. Our table model does not determine any mandatory functional regions like head, stub, stubhead, or body. We also do not bind a cell with a role (heading or data value). Content that forms an entry, label, or category name can be located anywhere in a table. Instead of determining a few popular table types in algorithms, we propose to specify their features via external rules expressed in our domain-specific language.

The methods for table interpretation are mainly knowledge-based. They try to bind text in tables with some external concepts, using the following techniques: extraction ontologies (Embley et al., 2005), data frames (Tijerino et al., 2005), databases with facts (classes and relations) automatically collected from the Web (Venetis et al., 2011), Linked Open Data like YAGO (Limaye et al., 2010), WIKITOLOGY (Mulwad et al., 2010), FREEBASE (Deng et al., 2013), DBPEDIA (Muñoz et al., 2014), or general purpose knowledge taxonomy, PROBASE (Wang et al., 2012). There are also several studies that propose to use contextual information that surrounds tables (Braunschweig, 2015; Govindaraju et al., 2013; Zhang, 2014; Yoshida et al., 2016).

These methods rely on textual content of tables and their context but neglect their style features. In practice tabular data can be expressed via not only text but also style (e. g. different fonts or colors in cells can also have interpretable meanings). In contrast to them, our tool enables to specify and use both layout and style features. Our work slightly concerns table interpretation. We believe that many of the mentioned techniques can allow to extend our tool for annotating and conceptualization of table content.

7.2. Tabular data extraction and transformation tools

The methods (Han et al., 2008; Langegger & Wöß, 2009; O'Connor et al., 2010; Mulwad et al., 2012; Fiorelli et al., 2015; Galkin et al., 2015; Ermilov & Ngomo, 2016) for generating linked data in RDF or OWL formats from spreadsheets or web tables include table analysis and interpretation. Some of them (Langegger & Wöß, 2009; O'Connor et al., 2010) work with various layouts of tables that differ from relational tables and propose domain-specific languages for specifying mappings of their data into structured representations. Langegger & Wöß (2009) present XLWrap, a spreadsheet-to-RDF wrapper where mappings are specified by XLWrap expressions. O'Connor et al. (2010) describe M^2, a mapping language for converting data from arbitrary tables presented in spreadsheets to OWL format. Our aim in contrast to the mentioned techniques consists in generating only relational tables.

Data transformation tools typically deal only with tables, which have a simple "matrix" layout, where there are no merged cells, hierarchical headers, or footnotes. Potter's Wheel (Raman & Hellerstein, 2001), Wrangler (Kandel et al., 2011), and OpenRefine[18] supply own domain-specific languages for cleaning messy values and reformatting tabular structures. HaExcel framework enables bidirectional mapping between a spreadsheet and a relational database (Cunha et al., 2009). MDSheet framework also implements a technique that automatically infers relational schemes from spreadsheets (Cunha et al., 2016). The methodology (Gulwani et al., 2012) based on programming by examples includes a domain-specific language and algorithms for synthesis of table layout and content transformations. One of the the latest work (Jin et al., 2017) develops Foofah, a system to synthesize data transformation programs by examples. In contrast to the mentioned techniques, we work with arbitrary tables with complex layout. Our tool transforms them to relational tables and ETL (Extract, Transform, Load) tools can be used in further data normalization and cleansing.

[18] `http://openrefine.org`

Hung et al. (2011) propose TranSheet, a spreadsheet-like formula language for specifying mappings between source spreadsheet data and a target schema. Their language is expressive and flexible for supporting practical spreadsheet-based data transformation in many cases. However, the mappings strictly determine correspondences between absolute cell addresses of source data and a target schema. Our language supports both absolute and relative cell addressing in comparison to TranSheet. Moreover, it allows formulating conditions for selecting cells through text and style features.

Barowy et al. (2015) introduce the extraction language Flare that extends regular expressions with geometric constructs. Interpretation of Flare constraints allows extracting and transforming tabular data from spreadsheets. Their tool, FlashRelate, synthesizes a program in Flare by a small set of examples (output relational tuples). The earlier study (Harris & Gulwani, 2011) presents TableProg, a language that can express practical transformations over tabular data in spreadsheets. It is also designed for end-user programming by examples (input and output tables). Unlike them, we express transformations as table analysis and interpretation rules.

Chen et al. (2016) focus on the problem of spreadsheet property (e. g. aggregation rows or hierarchical header) detection, identifying when a corresponding transformation program should be applied. They aim to convert any kind of spreadsheet data into relational tables. Their framework constructs trained property detectors based on rule-assisted active learning. It also uses crude user-provided rules on how a property can be detected for generating training data.

The recent works (de Vos et al., 2017; Cao et al., 2017) develop domain-specific solutions. de Vos et al. (2017) propose algorithms and accompanying software for automatic annotation of natural science spreadsheet tables. They combine structural properties of the tables (basic assumptions) and external vocabularies. Their algorithms implement a domain-specific set of classification rules and heuristics on these properties. The other tool (Cao et al., 2017) extracts RDF data from french government statistical spreadsheets and popu-

lates instances of their conceptual model. Both of these works limit ranges of processable tables by the considering domains.

To the best of our knowledge, there are no other domain-specific languages for tabular data transformation in terms of table understanding. Our language is based on the well-known terminology of Wang's model (Wang, 1996). In contrast to the existing mapping languages, we draw up this process as consecutive steps: role analysis, structural analysis, and interpretation.

7.3. Our previous work

We propose an approach to table understanding based on executing rules for table analysis and interpretation with a business rule engine (Shigarov, 2015b). We briefly introduce the preliminary version of our domain-specific rule language first in (Shigarov, 2015a). The prototype of our tool for canonicalization of arbitrary tables in spreadsheets is discussed in (Shigarov et al., 2016b). This paper combines and significantly expands our previous results. We explain in details our language in terms of the stages of table understanding. Moreover, this work revises and improves its syntax. We consider the original experiment application for transforming arbitrary tables of the same genre (government statistical tables). The performance evaluation has been done automatically for both role and structural analysis with the prepared ground-truth data first.

8. Conclusions

The work shows new possibilities in spreadsheet data transformation from arbitrary to relational tables based on rule-based programming for the table understanding stages.

The presented two-layered table object model combines the physical and logical table structure. Our model supports common layout and style spreadsheet features. Unlike others, it is not based on using functional cell regions. The functional items can be placed anywhere in a table. Therefore, the model

can represent any table type satisfying the spreadsheet layout constraints. Another difference from existed representations is that the model provides data provenance for recovered semantics.

CRL, our domain-specific language, enables to express table analysis and interpretation rules. In contrast to the existing mapping languages, we draw up this process as consecutive steps: role analysis, structural analysis, and interpretation. There are no other domain-specific languages for tabular data transformation in terms of table understanding. Although, a general-purpose rule language can also express table analysis and interpretation rules. However, our language allows focusing on the logic of table understanding without excessive details. CRL defines a set of the essential actions for table understanding stages.

TabbyXL, our tool for spreadsheet data canonicalization, implements both the model and the language. The experiment demonstrates that the tool can be used for developing programs for transformation of spreadsheet data into the relational form. One rule-set can process a wide range of tables of the same genre, e. g. government statistical websites. Our tool can be used for populating databases from arbitrary tables, which share common features. It also can serve as a part of data extraction from tables (e. g. in tabular document processing (Yang et al., 2017)).

The work focuses on rather table analysis than issues of interpretation. We only recover categories as sets of labels, without binding it with a taxonomy of concepts. The further work on table content conceptualization can overcome this limitation. Moreover, we observe that arbitrary tables can contain messy (e. g. non-standardized values or typos) and useless (e. g. aggregations or padding characters) data. It seems to be interesting for the further work to incorporate additional techniques of data cleansing in our tool. Another development direction is to integrate the presented results with tools for extracting tables from documents (e. g. untagged PDF documents (Shigarov et al., 2016a)) in end-to-end systems for table understanding.

9. Acknowledgments

This work was supported by Russian Foundation for Basic Research [grant number 16-57-44034]; Russian Academy of Sciences [program number I.33]; and Council for Grants of the President of Russian Federation [grant number NSh-8081.2016.9]. The presented software was developed with resources of the Shared Equipment Center of Integrated Information and Computing Network of Irkutsk Research and Educational Complex[19].

References

Abraham, R., & Erwig, M. (2007). UCheck: A spreadsheet type checker for end users. *Journal of Visual Languages & Computing*, *18*, 71–95. doi:10.1016/j.jvlc.2006.06.001.

Adelfio, M. D., & Samet, H. (2013). Schema extraction for tabular data on the web. *Proc. VLDB Endow.*, *6*, 421–432. doi:10.14778/2536336.2536343.

Barowy, D. W., Gulwani, S., Hart, T., & Zorn, B. (2015). FlashRelate: Extracting relational data from semi-structured spreadsheets using examples. *SIGPLAN Not.*, *50*, 218–228. doi:10.1145/2813885.2737952.

Braunschweig, K. (2015). *Recovering the Semantics of Tabular Web Data*. Ph.D. thesis Technischen Universitt Dresden Dresden, Germany.

Cafarella, M. J., Halevy, A., Wang, D. Z., Wu, E., & Zhang, Y. (2008). WebTables: Exploring the power of tables on the web. *Proc. VLDB Endow.*, *1*, 538–549. doi:10.14778/1453856.1453916.

Cao, T. D., Manolescu, I., & Tannier, X. (2017). Extracting linked data from statistic spreadsheets. In *Proc. Int. Workshop Semantic Big Data* (pp. 5:1–5:5). doi:10.1145/3066911.3066914.

[19] http://net.icc.ru

Chen, Z. (2016). *Information Extraction on Para-Relational Data*. Ph.D. thesis University of Michigan US.

Chen, Z., & Cafarella, M. (2013). Automatic web spreadsheet data extraction. In *Proc. 3rd Int. Workshop Semantic Search Over the Web* (pp. 1:1–1:8). doi:10.1145/2509908.2509909.

Chen, Z., & Cafarella, M. (2014). Integrating spreadsheet data via accurate and low-effort extraction. In *Proc. 20th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining* (pp. 1126–1135). doi:10.1145/2623330.2623617.

Chen, Z., Cafarella, M., Chen, J., Prevo, D., & Zhuang, J. (2013). Senbazuru: A prototype spreadsheet database management system. *Proc. VLDB Endow.*, *6*, 1202–1205. doi:10.14778/2536274.2536276.

Chen, Z., Rong, X., Dadiomov, S., Wesley, R., Xiao, G., Cory, D., Cafarella, M., & Mackinlay, J. (2016). *Spreadsheet Property Detection with Rule-assisted Active Learning*. Technical Report CSE-TR-601-16. URL: https://www.cse.umich.edu/techreports/cse/2016/CSE-TR-601-16.pdf.

Crestan, E., & Pantel, P. (2011). Web-scale table census and classification. In *Proc. 4th ACM Int. Conf. Web Search and Data Mining* (pp. 545–554). doi:10.1145/1935826.1935904.

Cunha, J., Erwig, M., Mendes, J., & Saraiva, J. (2016). Model inference for spreadsheets. *Autom Softw Eng*, *23*, 361–392. doi:10.1007/s10515-014-0167-x.

Cunha, J., Saraiva, J. a., & Visser, J. (2009). From spreadsheets to relational databases and back. In *Proc. ACM SIGPLAN Workshop Partial Evaluation and Program Manipulation* (pp. 179–188). doi:10.1145/1480945.1480972.

Deng, D., Jiang, Y., Li, G., Li, J., & Yu, C. (2013). Scalable column concept determination for web tables using large knowledge bases. *Proc. VLDB Endow.*, *6*, 1606–1617. doi:10.14778/2536258.2536271.

e Silva, A., Jorge, A., & Torgo, L. (2006). Design of an end-to-end method to extract information from tables. *Int. J. Document Analysis and Recognition*, *8*, 144–171. doi:10.1007/s10032-005-0001-x.

Eberius, J., Braunschweig, K., Hentsch, M., Thiele, M., Ahmadov, A., & Lehner, W. (2015). Building the dresden web table corpus: A classification approach. In *Proc. IEEE/ACM 2nd Int. Symposium on Big Data Computing* (pp. 41–50). doi:10.1109/BDC.2015.30.

Embley, D., Tao, C., & Liddle, S. (2005). Automating the extraction of data from HTML tables with unknown structure. *Data Knowl. Eng.*, *54*, 3–28. doi:10.1016/j.datak.2004.10.004.

Embley, D. W., Krishnamoorthy, M. S., Nagy, G., & Seth, S. (2016). Converting heterogeneous statistical tables on the web to searchable databases. *Int. J. Document Analysis and Recognition*, *19*, 119–138. doi:10.1007/s10032-016-0259-1.

Embley, D. W., Seth, S., & Nagy, G. (2014). Transforming web tables to a relational database. In *Proc. 22nd Int. Conf. Pattern Recognition* (pp. 2781–2786). doi:10.1109/ICPR.2014.479.

Ermilov, I., & Ngomo, A.-C. N. (2016). TAIPAN: Automatic property mapping for tabular data. In *Proc. 20th Int. Conf. Knowledge Engineering and Knowledge Management* (pp. 163–179). doi:10.1007/978-3-319-49004-5_11.

Fiorelli, M., Lorenzetti, T., Pazienza, M. T., Stellato, A., & Turbati, A. (2015). Sheet2RDF: a flexible and dynamic spreadsheet import&lifting framework for RDF. In *Proc. 28th Int. Conf. Industrial, Engineering and Other Applications of Applied Intelligent Systems* (pp. 131–140). doi:10.1007/978-3-319-19066-2_13.

Galkin, M., Mouromtsev, D., & Auer, S. (2015). Identifying web tables: Supporting a neglected type of content on the web. In *Proc. 6th Int.*

Conf. Knowledge Engineering and Semantic Web (pp. 48–62). doi:10.1007/978-3-319-24543-0_4.

Goto, K., Ohta, Y., Inakoshi, H., & Yugami, N. (2016). Extraction algorithms for hierarchical header structures from spreadsheets. In *Proc. Workshops of the EDBT/ICDT 2016 Joint Conference*. URL: http://ceur-ws.org/Vol-1558/paper27.pdf.

Govindaraju, V., Zhang, C., & Ré, C. (2013). Understanding tables in context using standard NLP toolkits. In *Proc. 51st Annual Meeting of the Association for Computational Linguistics* (pp. 658–664). volume 2: Short Papers.

Gulwani, S., Harris, W. R., & Singh, R. (2012). Spreadsheet data manipulation using examples. *Commun. ACM*, *55*, 97–105. doi:10.1145/2240236.2240260.

Han, L., Finin, T., Parr, C., Sachs, J., & Joshi, A. (2008). RDF123: From spreadsheets to RDF. In *Proc. 7th Int. Semantic Web Conf.* (pp. 451–466). doi:10.1007/978-3-540-88564-1_29.

Harris, W. R., & Gulwani, S. (2011). Spreadsheet table transformations from examples. *SIGPLAN Not.*, *46*, 317–328. doi:10.1145/1993316.1993536.

Hung, V., Benatallah, B., & Saint-Paul, R. (2011). Spreadsheet-based complex data transformation. In *Proc. 20th ACM Int. Conf. Information and Knowledge Management* (pp. 1749–1754). doi:10.1145/2063576.2063829.

Hurst, M. (2001). Layout and language: Challenges for table understanding on the web. In *Proc. 1st Int. Workshop Web Document Analysis* (pp. 27–30).

Jin, Z., Anderson, M. R., Cafarella, M., & Jagadish, H. V. (2017). Foofah: Transforming data by example. In *Proc. ACM Int. Conf. Management of Data* (pp. 683–698). doi:10.1145/3035918.3064034.

Kandel, S., Paepcke, A., Hellerstein, J., & Heer, J. (2011). Wrangler: Interactive visual specification of data transformation scripts. In *Proc. SIGCHI*

Conf. Human Factors in Computing Systems (pp. 3363–3372). doi:10.1145/1978942.1979444.

Kim, Y.-S., & Lee, K.-H. (2008). Extracting logical structures from HTML tables. *Computer Standards & Interfaces*, *30*, 296–308. doi:10.1016/j.csi.2007.08.006.

Koci, E., Thiele, M., Romero, O., & Lehner, W. (2016). A machine learning approach for layout inference in spreadsheets. In *Proc. 8th Int. Joint Conf. Knowledge Discovery, Knowledge Engineering and Knowledge Management* (pp. 77–88). doi:10.5220/0006052200770088.

Langegger, A., & Wöß, W. (2009). Xlwrap - querying and integrating arbitrary spreadsheets with SPARQL. In *Proc. 8th Int. Semantic Web Conf.* (pp. 359–374). doi:10.1007/978-3-642-04930-9_23.

Lautert, L. R., Scheidt, M. M., & Dorneles, C. F. (2013). Web table taxonomy and formalization. *SIGMOD Rec.*, *42*, 28–33. doi:10.1145/2536669.2536674.

Lehmberg, O., Ritze, D., Meusel, R., & Bizer, C. (2016). A large public corpus of web tables containing time and context metadata. In *Proc. 25th Int. Conf. Companion on World Wide Web* (pp. 75–76). doi:10.1145/2872518.2889386.

Limaye, G., Sarawagi, S., & Chakrabarti, S. (2010). Annotating and searching web tables using entities, types and relationships. *Proc. VLDB Endow.*, *3*, 1338–1347. doi:10.14778/1920841.1921005.

Mauro, N. D., Esposito, F., & Ferilli, S. (2013). Finding critical cells in web tables with SRL: Trying to uncover the devil's tease. In *Proc. 12th Int. Conf. on Document Analysis and Recognition* (pp. 882–886). doi:10.1109/ICDAR.2013.180.

Muñoz, E., Hogan, A., & Mileo, A. (2014). Using linked data to mine RDF from wikipedia's tables. In *Proc. 7th ACM Int. Conf. Web Search and Data Mining* (pp. 533–542). doi:10.1145/2556195.2556266.

Mulwad, V., Finin, T., & Joshi, A. (2012). A domain independent framework for extracting linked semantic data from tables. In *Search Computing: Broadening Web Search* (pp. 16–33). doi:10.1007/978-3-642-34213-4_2.

Mulwad, V., Finin, T., Syed, Z., & Joshi, A. (2010). Using linked data to interpret tables. In *Proc. 1st Int. Conf. Consuming Linked Data* (pp. 109–120). volume 665. URL: http://dl.acm.org/citation.cfm?id=2878947.2878957.

Nagy, G. (2012). Learning the characteristics of critical cells from web tables. In *Proc. 21st Int. Conf. Pattern Recognition* (pp. 1554–1557).

Nagy, G. (2016). TANGO-DocLab web tables from international statistical sites (Troy_200), 1, ID: Troy_200_1. URL: http://tc11.cvc.uab.es/datasets/Troy_200_1.

Nagy, G., Embley, D., & Seth, S. (2014). End-to-end conversion of HTML tables for populating a relational database. In *Proc. 11th IAPR Int. Workshop Document Analysis Systems* (pp. 222–226). doi:10.1109/DAS.2014.9.

Nagy, G., Embley, D. W., Krishnamoorthy, M., & Seth, S. (2015). Clustering header categories extracted from web tables. *Proc. SPIE*, *9402*, 94020M–94020M–12. doi:10.1117/12.2076209.

Nagy, G., & Seth, S. (2016). Table headers: An entrance to the data mine. In *Proc. 23rd Int. Conf. Pattern Recognition* (pp. 4065–4070). doi:10.1109/ICPR.2016.7900270.

O'Connor, M. J., Halaschek-Wiener, C., & Musen, M. A. (2010). Mapping master: A flexible approach for mapping spreadsheets to OWL. In *Proc. 9th Int. Semantic Web Conf., Part II* (pp. 194–208). doi:10.1007/978-3-642-17749-1_13.

Pivk, A., Cimiano, P., Sure, Y., Gams, M., Rajkovič, V., & Studer, R. (2007). Transforming arbitrary tables into logical form with TARTAR. *Data Knowl. Eng.*, *60*, 567–595. doi:10.1016/j.datak.2006.04.002.

Raman, V., & Hellerstein, J. M. (2001). Potter's wheel: An interactive data cleaning system. In *Proc. the 27th Int. Conf. Very Large Data Bases* (pp. 381–390). URL: http://dl.acm.org/citation.cfm?id=645927.672045.

Rastan, R., Paik, H.-y., Shepherd, J., & Haller, A. (2016). Automated table understanding using stub patterns. In *Proc. 21st Int. Conf. Database Systems for Advanced Applications, Part I* (pp. 533–548). doi:10.1007/978-3-319-32025-0_33.

Seth, S., & Nagy, G. (2013). Segmenting tables via indexing of value cells by table headers. In *Proc. 12th Int. Conf. Document Analysis and Recognition* (pp. 887–891). doi:10.1109/ICDAR.2013.181.

Shigarov, A. (2015a). Rule-based table analysis and interpretation. In *Proc. 21st Int. Conf. Information and Software Technologies* (pp. 175–186). doi:10.1007/978-3-319-24770-0_16.

Shigarov, A. (2015b). Table understanding using a rule engine. *Expert Systems with Applications*, *42*, 929–937. doi:10.1016/j.eswa.2014.08.045.

Shigarov, A. (2017). TabbyXL: Experiment data. Mendeley Data, v1. doi:10.17632/448jdx7gcr.1.

Shigarov, A., Mikhailov, A., & Altaev, A. (2016a). Configurable table structure recognition in untagged PDF documents. In *Proc. ACM Symposium on Document Engineering* (pp. 119–122). doi:10.1145/2960811.2967152.

Shigarov, A., Paramonov, V., Belykh, P., & Bondarev, A. (2016b). Rule-based canonicalization of arbitrary tables in spreadsheets. In *Proc. 22nd Int. Conf. Information and Software Technologies* (pp. 78–91). doi:10.1007/978-3-319-46254-7_7.

Tao, C., & Embley, D. W. (2009). Automatic hidden-web table interpretation, conceptualization, and semantic annotation. *Data Knowl. Eng.*, *68*, 683–703. doi:10.1016/j.datak.2009.02.010.

Tijerino, Y., Embley, D., Lonsdale, D., Ding, Y., & Nagy, G. (2005). Towards ontology generation from tables. *World Wide Web: Internet and Web Information Systems*, *8*, 261–285. doi:10.1007/s11280-005-0360-8.

Venetis, P., Halevy, A., Madhavan, J., Paşca, M., Shen, W., Wu, F., Miao, G., & Wu, C. (2011). Recovering semantics of tables on the web. *Proc. VLDB Endow.*, *4*, 528–538. doi:10.14778/2002938.2002939.

de Vos, M., Wielemaker, J., Rijgersberg, H., Schreiber, G., Wielinga, B., & Top, J. (2017). Combining information on structure and content to automatically annotate natural science spreadsheets. *International Journal of Human-Computer Studies*, *103*, 63–76. doi:10.1016/j.ijhcs.2017.02.006.

Wang, J., Wang, H., Wang, Z., & Zhu, K. Q. (2012). Understanding tables on the web. In *Proc. 31st Int. Conf. Conceptual Modeling* (pp. 141–155). doi:10.1007/978-3-642-34002-4_11.

Wang, X. (1996). *Tabular Abstraction, Editing, and Formatting*. Ph.D. thesis University of Waterloo Waterloo, Ontario, Canada.

Yang, S., Guo, J., & Wei, R. (2017). Semantic interoperability with heterogeneous information systems on the internet through automatic tabular document exchange. *Information Systems*, *69*, 195–217. doi:10.1016/j.is.2016.10.010.

Yoshida, M., Matsumoto, K., & Kita, K. (2016). Table topic models for hidden unit estimation. In *Proc. 12th Asia Information Retrieval Societies Conference* (pp. 302–307). doi:10.1007/978-3-319-48051-0_23.

Zhang, Z. (2014). Towards efficient and effective semantic table interpretation. In *Proc. 13th Int. Semantic Web Conference, Part I* (pp. 487–502). doi:10.1007/978-3-319-11964-9_31.