TabbyXL: Rule-Based Spreadsheet Data Extraction and Transformation

Alexey Shigarov[1,2(✉)], Vasiliy Khristyuk[1], Andrey Mikhailov[1], and Viacheslav Paramonov[1,2]

[1] Matrosov Institute for System Dynamics and Control Theory of SB RAS, 134 Lermontov st., Irkutsk, Russia
shigarov@icc.ru
[2] Institute of Mathematics, Economics and Informatics, Irkutsk State University, 20 Gagarin blvd., Irkutsk, Russia
http://cells.icc.ru

Abstract. This paper presents an approach to rule-based spreadsheet data extraction and transformation. We determine a table object model and domain-specific language of table analysis and interpretation rules. In contrast to the existing data transformation languages, we draw up this process as consecutive steps: role analysis, structural analysis, and interpretation. To the best of our knowledge, there are no languages for expressing rules for transforming tabular data into the relational form in terms of the table understanding. We also consider a tool for transforming spreadsheet data from arbitrary to relational tables. The performance evaluation has been done automatically for both (role and structural) stages of table analysis with the prepared ground-truth data. It shows high F-score from 95.82% to 99.04% for different recovered items in the existing dataset of 200 arbitrary tables of the same genre (government statistics).

Keywords: Data extraction · Data transformation · Table analysis · Rule-based programming · Spreadsheet

1 Introduction

A big volume of arbitrary tables (e.g. cross-tabulations, invoices, roadmaps, and data collection forms) circulates in spreadsheet-like formats. Mitlöhner et al. [28] estimate that about 10% of the resources in Open Data portals are labeled as CSV (Comma-Separated Values), a spreadsheet-like format. Barik et al. [2] extracted 0.25M unique spreadsheets from Common Crawl[1] archive. Chen and Cafarella [6] reported about 0.4M spreadsheets of ClueWeb09 Crawl[2] archive.

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

R. Damaševičius and G. Vasiljevienė (Eds.): ICIST 2019, CCIS 1078, pp. 59–75, 2019.
https://doi.org/10.1007/978-3-030-30275-7_6

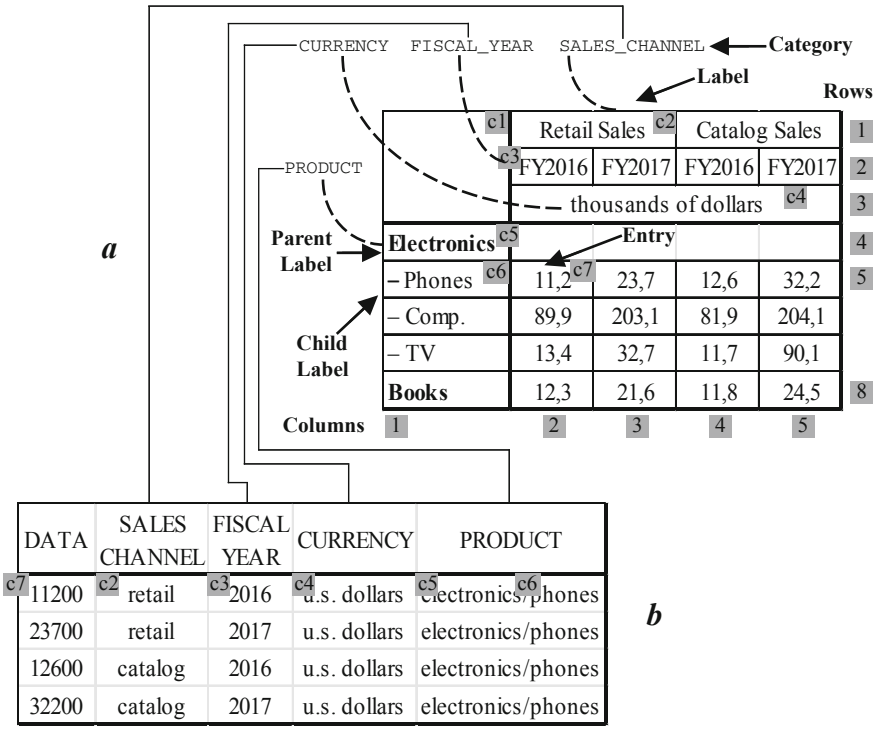

Fig. 1. A fragment of a source arbitrary spreadsheet table—*a*; a fragment of a target table in the relational (canonical) form generated from the source table—*b*.

Spreadsheets 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 on web pages can be easily converted to spreadsheet formats. The arbitrary spreadsheet 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.

Many of arbitrary spreadsheet tables are an instance of weakly-structured and non-standardized data (Fig. 1, *a*). Unlike relational tables (Fig. 1, *b*), they are not organized in a predefined manner. They lack explicit semantics required for high-level computer interpretation such as SQL queries. To be accessible for data analysis and visualization, their data need to be extracted, transformed, and loaded (ETL) into databases. Since arbitrary tables can have various complex cell layouts (structures), often the familiar industrial ETL-tools are not sufficient to populate automatically a database with their data.

Researchers and developers faced with the tasks of spreadsheet data integration often use general-purpose tools. They often offer their own implementations of the same tasks. In such cases, domain-specific tools can reduce the complexity

of software development in the domain of the spreadsheet data integration. This is especially important when a custom software for data extraction and transformation from heterogeneous arbitrary spreadsheet tables is implemented for a short time and with a lack of resources.

1.1 Related Work

There are several recent studies dealing with spreadsheet data converting. The tools [14–16,18,27,29,31] are devoted to issues of converting data presented in spreadsheets or web tables to RDF (Resource Description Framework) or OWL (Web Ontology Language) formats. The solutions for spreadsheet data extraction and transformation [1,3,17,19,22] are based on programming by examples. Some of the solutions also include own domain-specific languages: XLWrap [27], M^2 [31], TableProg [19], and Flare [3].

Hung et al. [20] propose TranSheet, a spreadsheet-like formula language for specifying mappings between source spreadsheet data and a target schema. Embley et al. [13] propose an algorithmic end-to-end solution for transforming "header-indexed" tables in CSV format to a relational form ("category tables") based on header indexing. Senbazuru, a spreadsheet database management system proposed by [5], provides extracting relational data from spreadsheets ("data frames"). HaExcel framework [10] enables migrating normalized data among spreadsheets and relational databases. MDSheet framework [8,9] also implements a technique that automatically infers relational schemes from spreadsheets. The framework [7] constructs trained spreadsheet property (e.g. aggregation rows or hierarchical header) detectors based on rule-assisted active learning.

DeExcelerator project [12] aims at the development of a framework for information extraction from partially structured documents such as spreadsheets and HTML tables. The framework exploits a set of heuristics based on features of tables published as Open Data. DeExcelerator works as a predefined pipeline without any user interaction. Koci et al. [23–25] expand the covered spectrum of spreadsheets. They proposes a machine learning approach for table layout inference [24] and TIRS, a heuristic-based framework for automatic table identification and reconstruction in spreadsheets [25]. Their recent paper [23] introduces a novel approach to recognition of the functional regions in single- and multi-table spreadsheets.

The recent papers [4,39] develop domain-specific solutions. The work [39] proposes algorithms and accompanying software for automatic annotation of natural science spreadsheet tables. The tool proposed in [4] extracts RDF data from French government statistical spreadsheets and populates instances of their conceptual model. The system CACheck [11] automatically detects and repairs "smelly" cell arrays by recovering their computational semantics. TaCLe [26] automatically identifies constraints (formulas and relations) in spreadsheets. The papers [41,42] suggest an approach to rule-based semantic extraction from tabular documents.

1.2 Contribution

Our work shows new possibilities in spreadsheet data transformation from arbitrary to relational tables based on rule-based programming for the following *table understanding* [21] stages:

- *Role analysis* extracts functional data items, entries (values) and labels (keys), from cell content.
- *Structural analysis* recovers relationships of functional data items (i.e. entry-label and label-label pairs).
- *Interpretation* binds recovered labels with categories (domains).

Our contribution consists in the following results:

- The two-layered table object model combines the physical (syntactic) and logical (semantic) table structure. Unlike others models, it is not based on using functional cell regions but determines that functional items can be placed anywhere in a table.
- The domain-specific language, CRL (Cells Rule Language), is intended for programming table analysis and interpretation rules. In contrast to the existing mapping languages, it expresses the spreadsheet data conversion in terms of table understanding.
- The tool for spreadsheet data extraction and transformation from an arbitrary (Fig. 1, *a*) to the relational (canonical) form (Fig. 1, *b*), TabbyXL[3], implements both the model and the language. Compared to the mentioned solutions it draws up this process as consecutive steps: role analysis, structural analysis, and interpretation.
- The CRL interpreter (CRL2J) provides the translation of CRL rules to Java code in the imperative style. It allows automatically generating Java source code from CRL rules and compile it to Java bytecode, and then runs generated Java programs. The interpreter uses CRL grammar implemented by ANTLR[4]. This ensures the correctness of CRL language grammar.
- The ruleset for transforming arbitrary tables of the same genre (government statistical websites) is implemented in three formats: CRL, DSLR (Drools[5]), and CLP (JESS[6]). The experimental results are reproduced and validated by using three different options: (i) CRL-to-Java translation; (ii) Drools rule engine; (iii) JESS rule engine. The results (recovered relational tables) are the same for all of these three options. This confirms the applicability of CRL language for expressing table analysis and interpretation rules.

This paper continues our series of works devoted the issues of table understanding in spreadsheets [33,34,37,38]. It combines and significantly expands our approach to table understanding based on executing rules for table analysis and interpretation with a business rule engine [34]. We briefly introduced

[3] https://github.com/tabbydoc/tabbyxl2.
[4] http://www.antlr.org.
[5] https://www.drools.org.
[6] http://www.jessrules.com.

the preliminary version of our domain-specific rule language first in [33]. The prototype of our tool for canonicalization of arbitrary tables in spreadsheets is discussed in [37]. This work extends the results presented in the paper [38] by adding the CRL interpreter for the CRL-to-Java translation, as well as by the implementation and validation of the ruleset by using the different options: CRL2J, Drools, and JESS.

The novelty of our current work consists in providing two rule-based ways to implement workflows of spreadsheet data extraction and transformation. In the first case, a ruleset for table analysis and interpretation is expressed in a general-purpose rule language and executed by a JSR-94-compatible rule engine (e.g. Drools or Jess). In the second case, our interpreter translates a ruleset expressed in CRL to Java source code that is complicated and executed by the Java development kit. This CRL-to-Java translation allows us to express rulesets without any instructions for management of the working memory such as updates of modified facts or blocks on the rule re-activation. The end-users can focus more on the logic of table analysis and interpretation than on the logic of the rule management and execution.

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 [40]. It includes two interrelated layers: *physical* (syntactic) represented by the collection of cells (Sect. 2.1) and *logical* (semantic) that consists of three collections of entries (values), labels (keys), and categories (concepts) (Sect. 2.2). We deliberately resort to the two-way references between the layers to provide convenient access to their objects in table analysis and interpretation rules.

2.1 Physical Layer

`Cell` object represents 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 features:

- *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 including: `name`, `color`, `size`, etc.; `horzAlignment` and `vertAlignment`—horizontal and vertical alignment; `bgColor` and `fgColor`—background and foreground colors; `leftBorder`, `topBorder`, `rightBorder`, and `bottomBorder`—border features; `rotation`—text rotation.
- *Content*: `text`—textual content, `indent`—indentation, and `type`—its literal data type (`numeric`, `date`, `string`, etc.).

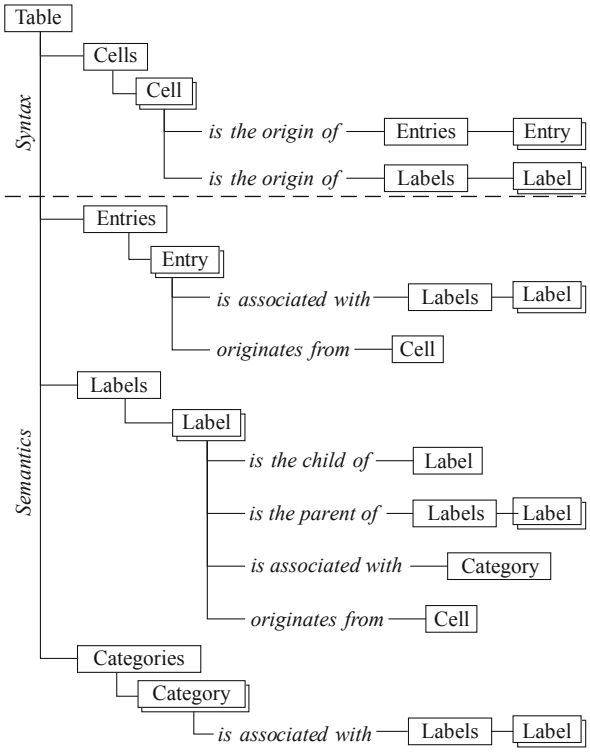

Fig. 2. Two-layered table object model.

- *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, Fig. 3 presents an initial state of some cells (`c1`,..., `c7`) shown in Fig. 1, *a*.

2.2 Logical Layer

`Entry` object serves as a representation 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).

```
c1=(cl=1,rt=1,cr=1,rb=3,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"
c4=(cl=2,rt=3,cr=5,rb=3,text="thousands of dollars")
c5=(cl=1,rt=4,cr=1,rb=4,text="Electronics")
c6=(cl=1,rt=5,cr=1,rb=5,text="- Phones")
c7=(cl=2,rt=5,cr=2,rb=5,text="11.2",type=NUMERIC)
```

Fig. 3. Some initial facts (cells) for the table shown in Fig. 1, *a*.

```
e1=(value="11200",labels={l1,l2,l3,l5},cell=c7)
l1=(value="retail",category=d1,cell=c2)
l2=(value="2016",category=d2,cell=c3)
l3=(value="u.s. dollars",category=d3,cell=c4)
l4=(value="electronics",children={l5,...}, category=d4,cell=c5)
l5=(value="phones",parent=l4,category=d4,cell=c6)
d1=(name="SALE CHANNEL",labels={l1,...})
d2=(name="FISCAL YEAR",labels={l2,...})
d3=(name="CURRENCY",labels={l3,...})
d4=(name="PRODUCT",labels={l4,l5,...})
```

Fig. 4. Some recovered facts (functional items) for the table shown in Fig. 1, *a*.

`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, Fig. 4 demonstrates a possible target state of the entry (`e1`), labels (`l1`,..., `l5`), and categories (`d1`,..., `d4`) recovered from the initial cells shown in Fig. 3. They can be presented as a tuple of a target relational Table 1, *b*.

3 CRL Language

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). Figure 5 demonstrates the grammar of CRL, our domain-specific language, in Extended Backus-Naur form. This grammar is also presented in ANTLR format[7].

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 #i
  when conditions
  then actions
end
```

[7] https://github.com/tabbydoc/tabbyxl2/blob/master/src/main/resources/crl_gram.g.

```
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 expr>
assignment = identifier ':' <a valid Java expr>
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 | group <EOL> {action}
merge      = 'merge' identifier 'with' identifier
split      = 'split' identifier
set text   = 'set text' <a Java string expr> 'to' identifier
set indent = 'set indent' <a Java integer expr> 'to' identifier
set mark   = 'set mark' <a Java string expr> 'to' identifier
new entry  = 'new entry' identifier ['as' <a Java string expr>]
new label  = 'new label' identifier ['as' <a Java string expr>]
add label  = 'add label' identifier | (<a Java string expr>
             'of' identifier | <a Java string expr>)
             'to' identifier
set parent   = 'set parent' identifier 'to' identifier
set category = 'set category' identifier | <a Java string expr>
               'to' identifier
group        = 'group' identifier 'with' identifier
identifier   = <a Java identifier>
```

Fig. 5. Grammar of CRL language.

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 var: constraints, assignments
entry var: constraints, assignments
label var: constraints, assignments
category var: constraints, assignments
```

The condition consists of three parts. In their order of occurrence, the first is a keyword that 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 satisfied to specified constraints: The first 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 split 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.

- `merge`, the action combines two adjacent cells when they share one border.
- `split`, the action divides a merged cell that spans n-tiles into n-cells. Each of the n-cells completely copies content and style from the merged cell and coordinates from the corresponding tile.
- `set text`, the action provides modifying textual content of a cell. Some string processing (e.g. regular expressions and string matching algorithms) implemented as Java-methods can be involved in the action.
- `set indent`, the action modifies an indentation of a 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 tags (marks) that can assist in both role and structural analysis.

- `set mark`, the action annotates a cell with a word or phrase. The assigned tag can substitute the corresponding constraints in subsequent rules. The typical practice is to set a tag to all cells, which play the same role or are located in the same table functional region. Thereafter, we can use these tags in subsequent rules instead of repeating constraints on cell location in the regions.
- `new entry`, the action generates an entry, using a specified cell as its origin. Usually, a value of the created entry is an expression obtained as a result of string processing for its textual content of the cell.
- `new label`, the action generates a label in a similar way.

3.4 Structural Analysis

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

- `add label`, the action binds an entry with an added label. A label can be specified as a value of a category indicated by its name.
- `set parent`, the action connects two labels as a parent and its child.

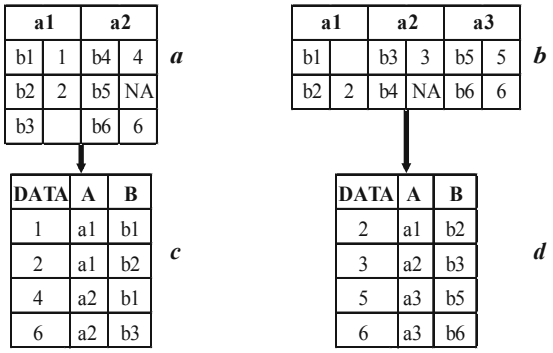

Fig. 6. Source tables—*a* and *b*; target canonicalized tables—*c* and *d*.

3.5 Interpretation

The stage includes actions for recovering label-category pairs.

- `set category`, the action associates a label with a category.
- `group`, the action places two labels in one group. 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. All labels of a group can be associated with only one category.

3.6 Illustrative Example

Figure 6 depicts an example of a transformational task that consists in converting tables similar to ones (*a* and *b*) into the relational form (*c* and *d*). These tables satisfy the following assumptions: $1, \ldots, n$ are entries, $a_1, \ldots, a_m$ are column

```
(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
```

Fig. 7. A reference ruleset for transforming the source tables (Fig. 6, *a*, *c*) to the target canonical forms (Fig. 6, *b*, *d*).

labels of the category A, $b_1, \ldots, b_k$ are row labels of the category B. Figure 7 presents a reference ruleset implemented for this task. It contains only 7 rules that are executed in the following order: (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.

4 Implementation with Options

TabbyXL implements both the presented table object model and the rule-based approach to spreadsheet data extraction and transformation. It can process data, using one of the following options:

- CRL2J-*option* automatically generates Java source code from CRL rules and compile it to Java bytecode, and then runs the generated program. This option requires that ruleset is implemented in CRL language. We use ANTLR, the parser generator, to implement the CRL-to-Java translator. This allows to parse CRL rules and to build their object model which is then translated to Java source code.
- Drools-*option* relies on Drools Expert rule engine. The rules can be expressed in DRL (the general-purpose rule language that is native for Drools) or in a dialect of CRL that is implemented as a domain-specific language (DSL) in corresponding of Drools requirements. In the last case, CRL rules presented in DSLR format are automatically translated into DRL format through the DSL-specification that defines CRL-to-DRL mappings. Unlike the pure CRL, this dialect supports DRL attributes in rule declarations.
- JESS-*option* executes a ruleset with JESS rule engine. This option requires that a ruleset is represented in the well-known CLP (CLIPS) format.

Moreover, the current version of TabbyXL supports rule engines that are compatible with Java Rule Engine API (JSR94[8]). Therefore, it is possible to use not only Drools or JESS, but also others rule engines supporting Java Rule Engine API, and to represent the rules in their native formats. Any case, TabbyXL builds an instance of the table object model from source spreadsheet data. The cells of this instance are asserted as facts into the working memory of a rule engine. Optionally, some user-defined categories specified in YAML format can also be loaded and presented as facts. The rule engine matches asserted facts against the rules. While rules are executed, the instance of the table object model is augmented by recovered facts (entries, labels, and categories). At the end of the transformation process, the instance is exported as a flat file database.

[8] https://www.jcp.org/ja/jsr/detail?id=94.

Table 1. Experiment results on the role and structural analysis stages.

Metrics	Role analysis		Structural analysis	
	Type of instances			
	Entries	Labels	Entry-label pairs	Label-label pairs
recall	0.9813 $\frac{16602}{16918}$	0.9965 $\frac{4842}{4859}$	0.9773 $\frac{34270}{35066}$	0.9389 $\frac{1951}{2078}$
precision	0.9996 $\frac{16602}{16609}$	0.9364 $\frac{4842}{5171}$	0.9965 $\frac{34270}{34389}$	0.9784 $\frac{1951}{1994}$
F-score	0.9904	0.9655	0.9868	0.9582

5 Experimental Results

The purpose of the experiment is to show a possibility of using our tool 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 ruleset for tables of the same genre, and (ii) evaluating the performance of the ruleset on a set of these tables.

We used Troy200 [30], the existing dataset of tables, for the performance evaluation. It contains 200 arbitrary tables as CSV files collected from 10 different sources of the same genre, government statistical websites predominantly presented in English language.

We designed and implemented a ruleset that transforms Troy200 arbitrary tables to the relational form, using two formats: CRL and CLP (JESS). We also prepared ground-truth data, including reference entries, labels, entry-label, and label-label pairs extracted from Troy200 tables. Their form is designed for human readability. Moreover, they are independent of the presence of critical cells in tables. The performance evaluation is based on comparing the ground-truth data with the tables generated by executing the presented ruleset for the experiment dataset.

All tables of the dataset were automatically transformed into the relational form, using TabbyXL with three different options: CRL2J, DROOLS, and JESS. The results are the same for all of these three options.

We used the standard metrics: *recall*, *precision*, and *F-score*, to evaluate our ruleset for both role and structural analysis. We adapted them as follows. When R is a set of instances in a target 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. These metrics were separately calculated for each type of instances (entries, labels, entry-label pairs, and label-label pairs).

The experiment results are shown in Table 1. Among 200 tables of the dataset, only 25 are processed with errors (1256 false negatives in 25 tables, and 498 false positives in 14 ones). Only one table is not processed. This results in 948 false

negative and 316 false positive errors, which amount to about 72% of all errors. In this case, entries are not recovered because they are not numeric as it is assumed in the used ruleset.

Table 2. Comparison of the running time by using the different options.

Running time of	CRL2J	Drools	JESS
Ruleset translation (t_1)	2108 ms	1711 ms	432 ms
Ruleset execution (t_2)	367 ms	1974 ms	4149 ms

Additionally, Table 2 presents a comparison of the running time for the ruleset translation and execution processes, using the implemented options. The ruleset translation time (t_1) is a time of translating an original ruleset into an executable form. The ruleset execution time (t_2) is a total time of executing the ruleset presented in the executable form that is required to process all 200 tables of the dataset. In the case of CRL2J option, t_1 is a time of parsing and compiling the original ruleset into a Java program, while t_2 is a time of executing the generated Java program. In the case of the use of a rule engine (Drools or JESS option), t_1 is a time of parsing the original ruleset and adding the result into a rule engine session. t_2 consists of a time of asserting initial facts into the working memory and a time of executing the ruleset by the rule engine. The results shown in Table 2 were obtained with the following computer: 3.2 GHz 4-core CPU and 8 GB RAM. The use of CRL2J option requires slightly more time for the translation of the ruleset. However, CRL2J is faster for the execution of the program (ruleset) compared to Drools or JESS rule engine.

All data and steps to reproduce the experiment results are publicly available as a dataset[9] [35]. It contains the following items: the relational tables obtained by TabbyXL; the ground-truth data; the mentioned CRL and CLP rulesets; the detailing of the performance evaluation. Some comparison of TabbyXL in its previous version with others solutions is discussed in the paper [38].

This experiment exemplifies the use of our language for developing task-specific rulesets. The performance evaluation confirms the applicability of the implemented ruleset in accomplishing the stated objectives of this application.

6 Conclusions and Further Work

The presented approach can be applied to develop software for extraction and transformation data of arbitrary spreadsheet tables. We expect TabbyXL to be useful in cases when data from a large number of tables appertaining to a few table types are required for populating a database.

We showed experimentally that rules for table analysis and interpretation can be expressed in a general-purpose rule-based language and executed with a

[9] https://data.mendeley.com/datasets/ydcr7mcrtp/3.

rule engine [34]. But only a few of its possibilities are sufficient for developing our rules. The exploration of these possibilities has led to development of CRL, a domain-specific rule language, which specializes in expressing only rules for table analysis and interpretation. Our language hides details which are inessential for us and allows to focus on the logic of table analysis and interpretation. There are two ways to use CRL rules. They can be executed with a rule engine (e.g. Drools or JESS) or be translated to programs presented in the imperative style (e.g. in Java language).

The experiment demonstrates that the tool can be used for developing programs for transformation of spreadsheet data into the relational form. One ruleset 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.

The work focuses rather on table analysis than on issues of interpretation. We only recover categories as sets of labels, without binding them with an external taxonomy of concepts (e.g. Linked Open Data). 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 direction of development is to integrate the presented results with the tools for extracting tables from documents (e.g. untagged PDF documents [32,36]) in end-to-end systems for the table understanding.

Acknowledgment. This work is supported by the Russian Science Foundation under Grant No.: 18-71-10001.

References

1. Astrakhantsev, N., Turdakov, D., Vassilieva, N.: Semi-automatic data extraction from tables. In: Selected Papers of the 15th All-Russian Scientific Conference on Digital Libraries: Advanced Methods and Technologies, Digital Collections, pp. 14–20 (2013)
2. Barik, T., Lubick, K., Smith, J., Slankas, J., Murphy-Hill, E.: Fuse: a reproducible, extendable, internet-scale corpus of spreadsheets. In: Proceedings of the 12th Working Conference on Mining Software Repositories, pp. 486–489. IEEE Press (2015). https://doi.org/10.1109/MSR.2015.70
3. Barowy, D.W., Gulwani, S., Hart, T., Zorn, B.: FlashRelate: extracting relational data from semi-structured spreadsheets using examples. SIGPLAN Not. **50**(6), 218–228 (2015). https://doi.org/10.1145/2813885.2737952
4. Cao, T.D., Manolescu, I., Tannier, X.: Extracting linked data from statistic spreadsheets. In: Proceedings of the International Workshop on Semantic Big Data, pp. 5:1–5:5 (2017). https://doi.org/10.1145/3066911.3066914
5. Chen, Z.: Information extraction on para-relational data. Ph.D. thesis, University of Michigan, US (2016)
6. Chen, Z., Cafarella, M.: Automatic web spreadsheet data extraction. In: Proceedings of the 3rd International Workshop on Semantic Search Over the Web, pp. 1:1–1:8 (2013). https://doi.org/10.1145/2509908.2509909

7. Chen, Z., et al.: Spreadsheet property detection with rule-assisted active learning. Technical report CSE-TR-601-16 (2016). https://www.cse.umich.edu/techreports/cse/2016/CSE-TR-601-16.pdf
8. Cunha, J., Erwig, M., Mendes, J., Saraiva, J.: Model inference for spreadsheets. Autom. Softw. Eng. **23**(3), 361–392 (2016). https://doi.org/10.1007/s10515-014-0167-x
9. Cunha, J., Fernandes, J.P., Mendes, J., Saraiva, J.: Spreadsheet engineering. In: Zsók, V., Horváth, Z., Csató, L. (eds.) CEFP 2013. LNCS, vol. 8606, pp. 246–299. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15940-9_6
10. Cunha, J., Saraiva, J.a., Visser, J.: From spreadsheets to relational databases and back. In: Proceedings of the ACM SIGPLAN Workshop Partial Evaluation and Program Manipulation, pp. 179–188 (2009). https://doi.org/10.1145/1480945.1480972
11. Dou, W., Xu, C., Cheung, S.C., Wei, J.: CACheck: detecting and repairing cell arrays in spreadsheets. IEEE Trans. Software Eng. **43**(3), 226–251 (2017). https://doi.org/10.1109/TSE.2016.2584059
12. Eberius, J., Werner, C., Thiele, M., Braunschweig, K., Dannecker, L., Lehner, W.: DeExcelerator: a framework for extracting relational data from partially structured documents. In: Proceedings of the 22nd ACM International Conference on Information & Knowledge Management, pp. 2477–2480 (2013). https://doi.org/10.1145/2505515.2508210. http://doi.acm.org/10.1145/2505515.2508210
13. Embley, D.W., Krishnamoorthy, M.S., Nagy, G., Seth, S.: Converting heterogeneous statistical tables on the web to searchable databases. IJDAR **19**(2), 119–138 (2016). https://doi.org/10.1007/s10032-016-0259-1
14. Ermilov, I., Ngomo, A.-C.N.: TAIPAN: automatic property mapping for tabular data. In: Blomqvist, E., Ciancarini, P., Poggi, F., Vitali, F. (eds.) EKAW 2016. LNCS (LNAI), vol. 10024, pp. 163–179. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49004-5_11
15. Fiorelli, M., Lorenzetti, T., Pazienza, M.T., Stellato, A., Turbati, A.: Sheet2RDF: a flexible and dynamic spreadsheet import&lifting framework for RDF. In: Ali, M., Kwon, Y., Lee, C.H., Kim, J., Kim, Y. (eds.) IEA/AIE 2015. LNCS, vol. 9101, pp. 131–140. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19066-2_13
16. Galkin, M., Mouromtsev, D., Auer, S.: Identifying web tables: supporting a neglected type of content on the web. In: Klinov, P., Mouromtsev, D. (eds.) KESW 2015. CCIS, vol. 518, pp. 48–62. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24543-0_4
17. Gulwani, S., Harris, W.R., Singh, R.: Spreadsheet data manipulation using examples. Commun. ACM **55**(8), 97–105 (2012). https://doi.org/10.1145/2240236.2240260
18. Han, L., Finin, T., Parr, C., Sachs, J., Joshi, A.: RDF123: from spreadsheets to RDF. In: Sheth, A., et al. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 451–466. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88564-1_29
19. Harris, W.R., Gulwani, S.: Spreadsheet table transformations from examples. SIGPLAN Not. **46**(6), 317–328 (2011). https://doi.org/10.1145/1993316.1993536
20. Hung, V., Benatallah, B., Saint-Paul, R.: Spreadsheet-based complex data transformation. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, pp. 1749–1754 (2011). https://doi.org/10.1145/2063576.2063829
21. Hurst, M.: Layout and language: challenges for table understanding on the web. In: Proceedings of the 1st International Workshop on Web Document Analysis, pp. 27–30 (2001)

22. Jin, Z., Anderson, M.R., Cafarella, M., Jagadish, H.V.: Foofah: transforming data by example. In: Proceedings of the ACM International Conference on Management of Data, pp. 683–698 (2017). https://doi.org/10.1145/3035918.3064034
23. Koci, E., Thiele, M., Lehner, W., Romero, O.: Table recognition in spreadsheets via a graph representation. In: 13th IAPR International Workshop on Document Analysis Systems, pp. 139–144 (2018). https://doi.org/10.1109/DAS.2018.48
24. Koci, E., Thiele, M., Romero, O., Lehner, W.: A machine learning approach for layout inference in spreadsheets. In: Proceedings of the 8th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, pp. 77–88 (2016). https://doi.org/10.5220/0006052200770088
25. Koci, E., Thiele, M., Romero, O., Lehner, W.: Table identification and reconstruction in spreadsheets. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 527–541. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_33
26. Kolb, S., Paramonov, S., Guns, T., De Raedt, L.: Learning constraints in spreadsheets and tabular data. Mach. Learn. **106**(9), 1441–1468 (2017). https://doi.org/10.1007/s10994-017-5640-x
27. Langegger, A., Wöß, W.: XLWrap – querying and integrating arbitrary spreadsheets with SPARQL. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 359–374. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04930-9_23
28. Mitlöhner, J., Neumaier, S., Umbrich, J., Polleres, A.: Characteristics of open data CSV files. In: 2nd International Conference on Open and Big Data, pp. 72–79 (2016). https://doi.org/10.1109/OBD.2016.18
29. Mulwad, V., Finin, T., Joshi, A.: A domain independent framework for extracting linked semantic data from tables. In: Ceri, S., Brambilla, M. (eds.) Search Computing. LNCS, vol. 7538, pp. 16–33. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34213-4_2
30. Nagy, G.: TANGO-DocLab web tables from international statistical sites (Troy_200), 1, ID: Troy_200_1 (2016). http://tc11.cvc.uab.es/datasets/Troy_200_1
31. O'Connor, M.J., Halaschek-Wiener, C., Musen, M.A.: Mapping master: a flexible approach for mapping spreadsheets to OWL. In: Patel-Schneider, P.F., et al. (eds.) ISWC 2010. LNCS, vol. 6497, pp. 194–208. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17749-1_13
32. Shigarov, A., Altaev, A., Mikhailov, A., Paramonov, V., Cherkashin, E.: TabbyPDF: web-based system for PDF table extraction. In: Damaševičius, R., Vasiljevienė, G. (eds.) ICIST 2018. CCIS, vol. 920, pp. 257–269. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99972-2_20
33. Shigarov, A.: Rule-based table analysis and interpretation. In: Dregvaite, G., Damasevicius, R. (eds.) ICIST 2015. CCIS, vol. 538, pp. 175–186. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24770-0_16
34. Shigarov, A.: Table understanding using a rule engine. Expert Syst. Appl. **42**(2), 929–937 (2015). https://doi.org/10.1016/j.eswa.2014.08.045
35. Shigarov, A., Khristyuk, V.: TabbyXL2: experiment data. Mendeley Data, v2 (2018). https://doi.org/10.17632/ydcr7mcrtp.2
36. Shigarov, A., Mikhailov, A., Altaev, A.: Configurable table structure recognition in untagged PDF documents. In: Proceedings of the ACM Symposium on Document Engineering, pp. 119–122 (2016). https://doi.org/10.1145/2960811.2967152

37. Shigarov, A.O., Paramonov, V.V., Belykh, P.V., Bondarev, A.I.: Rule-based canonicalization of arbitrary tables in spreadsheets. In: Dregvaite, G., Damasevicius, R. (eds.) ICIST 2016. CCIS, vol. 639, pp. 78–91. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46254-7_7
38. Shigarov, A.O., Mikhailov, A.A.: Rule-based spreadsheet data transformation from arbitrary to relational tables. Inf. Syst. **71**, 123–136 (2017). https://doi.org/10.1016/j.is.2017.08.004
39. de Vos, M., Wielemaker, J., Rijgersberg, H., Schreiber, G., Wielinga, B., Top, J.: Combining information on structure and content to automatically annotate natural science spreadsheets. Int. J. Hum. Comput. Stud. **103**, 63–76 (2017). https://doi.org/10.1016/j.ijhcs.2017.02.006
40. Wang, X.: Tabular abstraction, editing, and formatting. Ph.D. thesis, University of Waterloo, Waterloo, Ontario, Canada (1996)
41. Yang, S., Guo, J., Wei, R.: Semantic interoperability with heterogeneous information systems on the internet through automatic tabular document exchange. Inf. Syst. **69**, 195–217 (2017). https://doi.org/10.1016/j.is.2016.10.010
42. Yang, S., Wei, R., Shigarov, A.: Semantic interoperability for electronic business through a novel cross-context semantic document exchange approach. In: Proceedings of the ACM Symposium on Document Engineering, pp. 28:1–28:10 (2018). https://doi.org/10.1145/3209280.3209523