Table Layout Regular Expression - Layex

Romuald Rousseau

2024-03-02

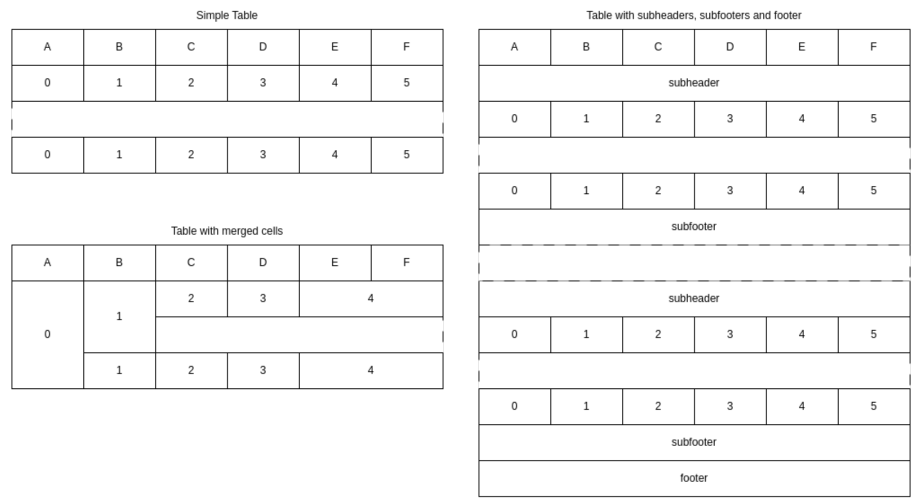
Abstract

In the modern landscape of data presentation, tables serve as a ubiquitous tool for organizing and conveying information efficiently. Whether in the structured presentation of scientific findings or the widespread use of spreadsheets in corporate environments, tables play a pivotal role in facilitating data interpretation. Consequently, the extraction of valuable insights encapsulated within these tables becomes paramount in any data pipeline process. This white paper introduces a novel mechanism designed to streamline the extraction of data from tables, particularly those with intricate layouts. Through the construction of a regular language customized to tabular representation, it aims to enhance efficiency and accuracy in data extraction processes, ultimately empowering organizations to unlock the full potential of their tabular data assets.

1 Introduction

Tables serve as a fundamental tool for organizing and presenting information in a structured and comprehensible manner. However, the diverse formats and complexities inherent in tables often pose challenges in effectively extracting relevant data. Recognizing the need for a streamlined solution, this white paper introduces a mechanism designed to decipher the intricate structures of tables.By offering an efficient method to characterize complex table layouts, this mechanism enables the accurate detection and extraction of various components and data points. This solution aims to address the inherent complexities associated with table processing, ultimately enhancing data extraction capabilities for improved decision-making and analysis. The following figure shows few examples of table layouts:

Figure 1: Various table layouts



2 About other research

A lot of research is made to extract data from table using machine learning and probabilistic approaches. While those approaches give very good result, they often failed in the consistence of the results. Our approach is purely deterministic, therefore the data extraction is very consistent and predictable. To be noted, that our approach can be used jointly with machine learning, and our compact way to describe table layout can be used to quickly create training sets and/or labels.

3 Data Used and Statistics

The following table shows the number of documents used to experiement and the number od documents that this system since helped to ingest:

Languages	Experiment Docs	Prod Docs	Prod Records
8	445	4,777	27,698,600

4 Anatomy of a table

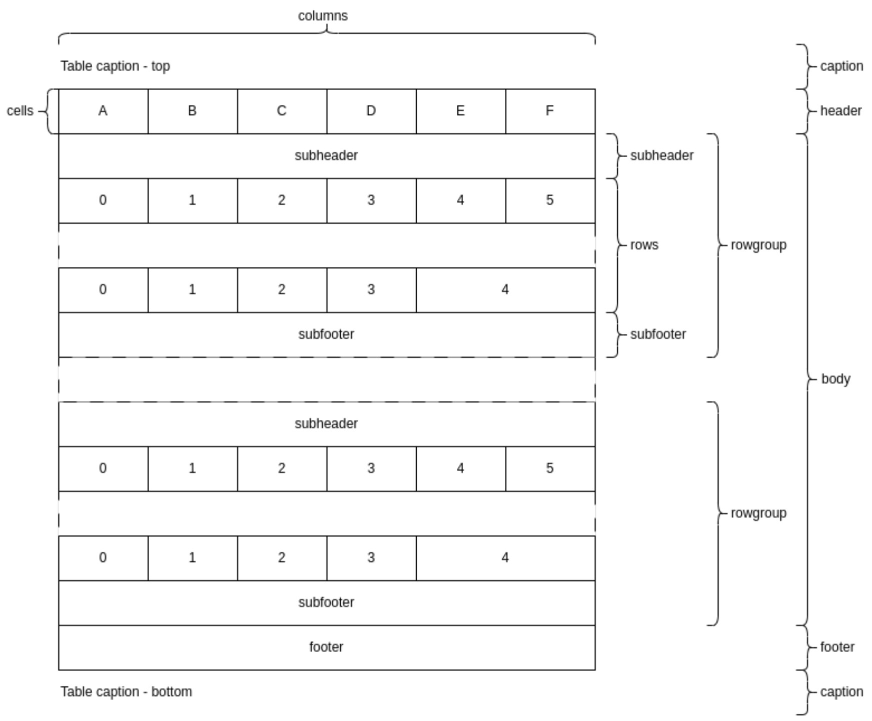
By analyzing different layouts based on hundred of documents, a table can be categorized in different basic components:

- Columns, Rows, Cells.
- Captions, Header, Footer, Body.

- Row Groups, Sub-headers, Sub-footers.

All these components are structurally linked to each others and form repeated patterns. The basic relation between the components of a table follows the direction of reading, left top right, top to bottom for most western languages.The following figure shows a graphical overview of those components and their relation to each others:

Figure 2: Ananomy of a table



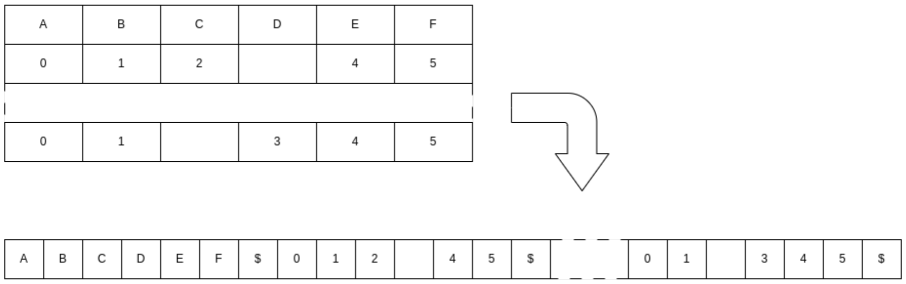
4.1 More formally

A table is basically a grid (rows and columns) of cells. Cell is the smallest component of a table. A cell can contain a value, be empty or be merged with its neighbors. If we look closely, a table can be represented as a stream of cells ordered through the direction of reading and separated by end of row elements. Below an example of a simple table represented as a stream of cells, where:

- \$ is an end of row.
- [] is an empty content of a cell.
- [A-F] is a content of a header cell.

- [0-5] is a content of a row cell

Figure 3: Stream over a Table



5 Formally

5.1 Definition of a Table

The collection of table T over an alphabet Σ is defined recursively as follows:

- The empty table Ø is a table
- The end of row \$, the singleton {\$} is a table
- For each cell with a content $c \in \Sigma$, the singleton $\{c\}$ is a table
- If A is a table, A* (Kleene star) is a table.
- If A and B are tables, then $A \cdot B$ (concatenation) is a table

5.2 Definition of a Reading Direction

A reading direction is defined by a tuple of a directions. Directions are defined as follows:

- Vertical direction when cells are vertically arranged. We can have respectively 2 vertical directions top (T) to bottom (B) or bottom (B) to top (T) that we notes respectively TB or BT.
- Horizontal direction when cells are horizontally arranged. We can have respectively 2 horizontal directions left (L) to right (R) or right (R) to left (L) that we notes respectively LR or RL.
- The first element of the tuple is the primary direction or line direction and the second element is the secondary direction or character direction.

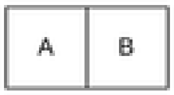
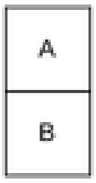
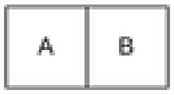
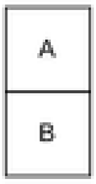
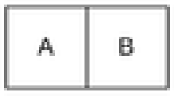
Few examples:

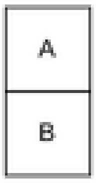
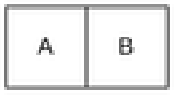
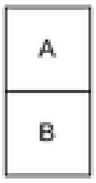
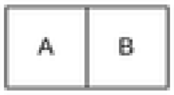
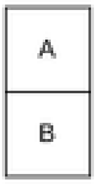
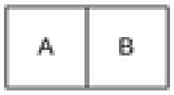
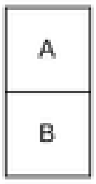
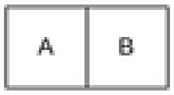
- TB-LR defines the English direction of reading when we read lines from top to bottom and characters from left to right. This direction is sometimes called Gutenberg direction.
- TB-RL defines the Arab direction of reading when we read lines from top to bottom and characters from right to left
- RL-TB defines the traditional Chinese direction of reading when we read lines from right to left and characters top to bottom

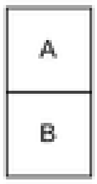
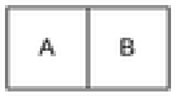
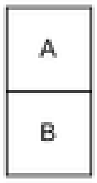
5.3 Definition of a Stream over a Table and a Reading Direction

The collection of stream S over the table T and a reading direction R is defined by the following transformations:

Table 1: Stream Transformation

Reading Direction	Table	Stream
TB-LR		$A \cdot B$
		$A \cdot \{\$\} \cdot B$
TB-RL		$B \cdot A$
		$A \cdot \{\$\} \cdot B$
BT-LR		$A \cdot B$

		$B \cdot \{\$\} \cdot A$
BT-RL		$B \cdot A$
		$B \cdot \{\$\} \cdot A$
TB-LR		$A \cdot B$
		$A \cdot \{\$\} \cdot B$
TB-RL		$B \cdot A$
		$A \cdot \{\$\} \cdot B$
BT-LR		$A \cdot B$

		$B \cdot \{\$\} \cdot A$
BT-RL		$B \cdot A$
		$B \cdot \{\$\} \cdot A$

5.4 Merged cells

Cells can be merged with their neighbors however we didn't define this concept in our formalism above. It is because, we can avoid merged cells by simply copy the content of the cell vertically or horizontally. Formally and depending of the reading direction:

- A cell merged of N cells in the character direction with the content $c \in \Sigma$ can be transformed as a concatenation $\{c\}^N$
- A cell merged of N cells in the line direction with the content $c \in \Sigma$ can be transformed as a concatenation $(\{c\} \cdot \{\$\})^N$

6 Regular Language of a Stream over a Table and a Reading Direction

6.1 Regular Language formal definition

The collection of regular languages over an alphabet Σ is defined recursively as follows:

- The empty language Ø is a regular language.
- For each $a \in \Sigma$(a belongs to Σ), the singleton language $\{a\}$ is a regular language.
- If A is a regular language, A* (Kleene star) is a regular language. Due to this, the empty string language $\{\epsilon\}$ is also regular.

- If A and B are regular languages, then $A \cup B$ (union) and $A \cdot B$ (concatenation) are regular languages.
- No other languages over Σ are regular.

6.2 Regular Language of a Stream over a Table and a Reading Direction

Let define the alphabet $\Sigma = \{s, e, v\}$ as follows:

- s, space, is an empty cell (containing spaces or nothing)
- e, entity, is a cell containing a string of characters matching a regex $r \in R$
  - A number ([0-9]+)
  - A date ([0-9]4-[0-9]2-[0-9]2)
  - ...
- v, value, is a cell containing a string that is not an entity or a space

By construction, a stream over a Table over the alphabet Σ and a Reading Direction is a regular language.

6.3 Regular Expression

Regular Expression describes a regular language that can be recognized by a finite automaton. A stream over a table and a reading direction is a regular language and therefore can be describes by a regular expression. It means a regular expression can match and extract patterns from a table after transformation into a stream over the given table and a given reading direction.

7 Table Layout Regular Expression or Layex

Table Layout Regular Expression or Layex is a syntax implementing a regular expression describing the regular language of a stream over a table and a reading direction. Layex syntax is similar to the commonly used regex syntax.

7.1 Boolean "or"

A vertical bar separates alternatives. For example, gray—grey can match "gray" or "grey".

7.2 Grouping

Parentheses are used to define the scope and precedence of the operators (among other uses). For example, gray—grey and gr(a—e)y are equivalent patterns which both describe the set of "gray" or "grey".

7.3 Quantification

A quantifier after an element (such as a token, character, or group) specifies how many times the preceding element is allowed to repeat. The most common quantifiers are the question mark ?, the asterisk * (derived from the Kleene star), and the plus sign + (Kleene plus).

Table 2: Quantification Symbol

?	The question mark indicates zero or one occurrences of the preceding element. For example, colou?r matches both "color" and "colour".
*	The asterisk indicates zero or more occurrences of the preceding element. For example, ab*c matches "ac", "abc", "abbc", "abbbc", and so on.
+	The plus sign indicates one or more occurrences of the preceding element. For example, ab+c matches "abc", "abbc", "abbbc", and so on, but not "ac".
{n}	The preceding item is matched exactly *n* times.
{min,}	The preceding item is matched *min* or more times.
{,max}	The preceding item is matched up to *max* times.
{min,max}	The preceding item is matched at least *min* times, but not more than *max* times.

7.4 Wildcard

The wildcard . matches any character. For example, a.b matches any string that contains an "a", and then any character and then "b". a.*b matches any string that contains an "a", and then the character "b" at some later point.

7.5 Negation

The upper case of an element means all but this element; S means everything except space.

8 General Algorithm

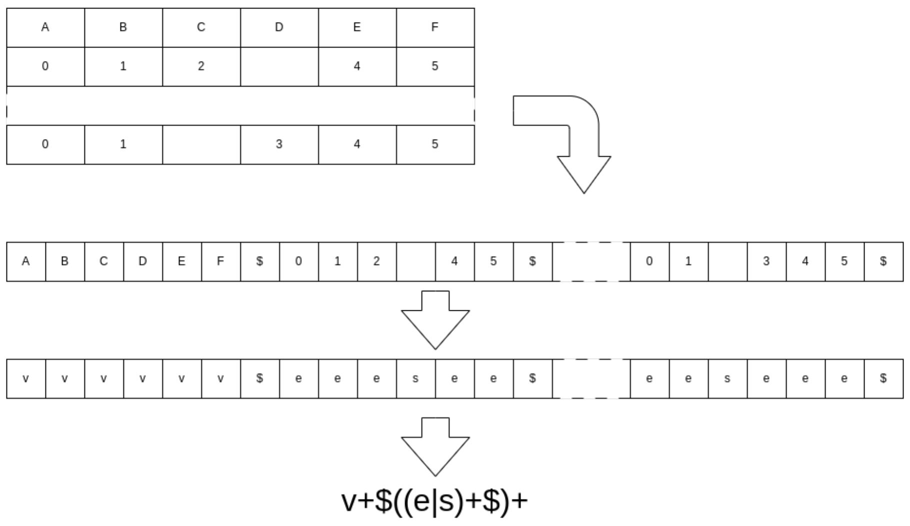
Below the outline of the algorithm in a pseudo Python language:

Listing 1: Layex Algorithm

```
import re

from openpyxl import load_workbook


def match_entity(v, R):
    for r in R:
    if re.compile(r).match(v) is not None:
        return True
    return False


def cell_to_alphabet(cell, R):
    if cell.value.isspace():
        return "s"
    elif match_entity(cell.value, R):
        return "e"
    else:
        return "v"


def table_to_stream(rows, R):
    for row in rows:
    for cell in row:
        if cell.ismerged():
        for n in cell.merged_horiz_range():
            yield cell_to_alphabet(cell, R)
        else:
        yield cell_to_alphabet(cell, R)
    yield "$"


R = {r'[0-9]'}
reg = re.compile(r'v+$((e|s)+$)+')

wb = load_workbook(
        filename='example.xlsx',
        read_only=True
    )
ws = wb.active

# Supposing re package works on iterable
reg.match(table_to_stream(ws.rows, R))
```

Below the basic steps of the algorithm:

Figure 4: Layex Algorithm


A
B
C
D
E
F
0
1
2
4
5
0
1
3
4
5
A B C D E F $ 0 1 2 4 5 $ 0 1 3 4 5 $
v v v v v v $ e e e s e e $ e e s e e e $
v+$((e|s)+$)+

9 Results and Statistics

The following tables give some statitics and the number of layex we wrote to ingest all those documents. The ratio number of layex verses the quantity of docs parsed demonstrates the flexibility and robusteness of this approach.

Avg Records per Docs	Max Records per Docs	Layex
5,798	1,048,575	15

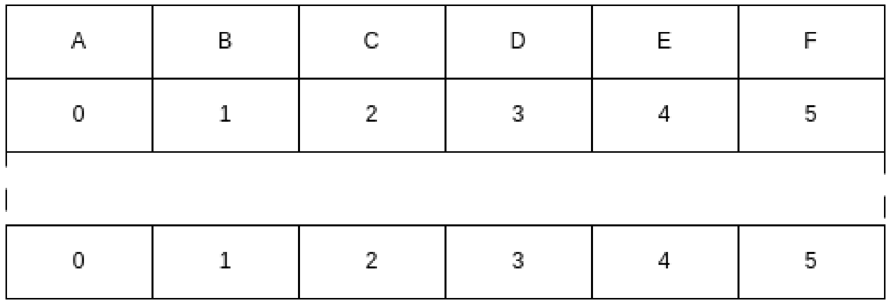
10 Few examples

Layex are a very efficient way to describe complex table layout and therefore detect them and extract the different components and data from them. Below are few examples.

10.1 Simple table

The first example is a simple table with a header and few rows:

Figure 5: Simple Table



A	B	C	D	E	F
0	1	2	3	4	5
0	1	2	3	4	5

This simple table can be described with the layex:

$$(. + \$)((. + \$)*)$$

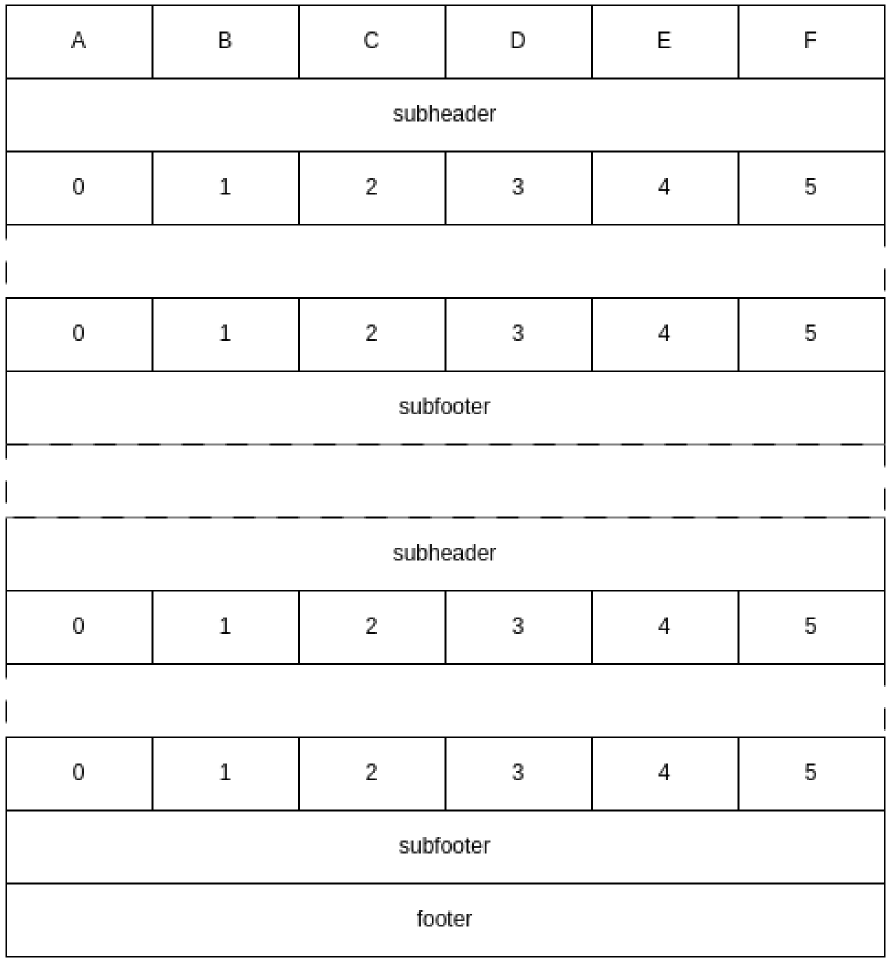
Where,

- $(. + \$)$ matches the header
    - $((. + \$)*)$ matches the body
        - $(. + \$)*$ matches the rows
            - $(. + \$)$ matches a row

10.2 Complex table

The first example is a complex table with a header, sub-header, sub-footer and footer:

Figure 6: Complex Table



This simple table can be described with the layex:

$$(v\$)(v+\$)((v\$)(.+\$)*(v\$))*(.+\$)(v\$)$$

Where,

- $(v\$)$, matches the top caption
- $(v+\$)$ matches the header
- $((v\$)(.+\$)*(v\$))*$ matches the body
  - $((v\$)(.+\$)*(v\$))$ matches a row-group
    - $(v\$)$ matches the sub-header

    - $(.+\$)*$ matches the rows
        - $(.+\$)$ matches a row
    - $(v\$)$ matches the sub-footer

- $(.+\$)$ matches the footer
- $(v\$)$ matches the bottom caption

11 References

- Regular Expression
- theory-of-computation-automata-tutorials
- TableExtraction-irj06.pdf

12 Implementation

Java - github