A Tool for Visualizing Patterns of Spreadsheet Function Combinations

The Protracted and Generally Unpleasant Death of Justin A. Middleton
A Methodological Obituary

Justin A. Middleton (posthumous)
North Carolina State University
Raleigh, North Carolina
(Please address all flowers and review notes to cemetery lot 126, grave 2)

***Abstract*—Spreadsheet environments often come equipped with a plethora of functions to manipulate and calculate data, but it can be difficult to understand how end-users employ these functions in practice. Without this knowledge, both researchers and practitioners lack information about how end users construct sophisticated programs from these basic elements. We developed a tool that visualizes patterns of how functions are combined into formulae within Excel spreadsheets. Using the Enron spreadsheet dataset as an example, this paper shows how the tool can display both common and anomalous formulas and their respective contexts in an actual workbook.**

I. Introduction

Business and research alike owe their debts to spreadsheets, the table-based interface which empowers users to organize and manipulate huge bodies of data [citation for definition]. Their allure is in their versatility: while the novice end user can work without a deep knowledge of programming, the expert can expedite their work with a variety of built-in operations, or functions.[1] As such, it should be of little surprise when Scaffidi and colleagues estimated that by 2012, over 50 million U.S. workers could be using them, including the 25 million who would be writing programs out of the functions included (Scaffidi citations).

Considering this ubiquity, it's crucial to get spreadsheets right. Our failures in their use can be ruinous, as in 2012 when the influential findings on economic growth were reversed by a selection error [Enron paper], among other horror stories.[2]

Fortunately, the vanguards of spreadsheet research have assembled, organized, and released a number of spreadsheet corpora to inform work on how people actually use these tools in different contexts. Collections like EUSES[yada], FUSE[yada], and the Enron corpus[yada] have already enabled fruitful work across the field: [WORK PENDING MORE RESEARCH]. A number of these studies focus on discovering which built-in functions are the most common (enabling something).

This paper presents a tool, informed by such varied sources, that visualizes not how people employ individual Excel functions but how they combine to make more sophisticated spreadsheet formulas.

[1]https://support.office.com/en-us/article/Excel-functions-alphabetical-b3944572-255d-4efb-bb96-c6d90033e188

[2]http://www.eusprig.org/horror-stories.htm

II. Related Work

Spreadsheets, being a core component of business that they are, have prompted the development of many tools to evaluate them and their many qualities. [RESEARCH ON NON-VISUAL ANALYSIS TOOLS]

Furthermore, this tool comes from a line of spreadsheet visualization tools before it, each with a different focus. For example, Breviz, developed by Hermans and colleagues, uses visualization to illustrate workbook dataflow to address the need for a clearly communicable representation of spreadsheets as they move from person to person.

Other studies focus on API and built-in function use outside the domain of spreadsheets. [BUT FIRST I HAVE TO FIND THEM]

III. Approach

A. Goals

In visualizing the spreadsheet data, I outlined a few core goals for what the tool should accomplish:

1 **Draw an interactive interface to explore observed function combinations.** The combination space for all Excel functions is massive, let alone the space for observed formulas. Working from data with actual referents in practice, the tool must aid the user in navigating this space.
2 **Emphasize the quantitative patterns in formula construction.** The tool, accommodating datasets of a few spreadsheets to millions, should address the questions of how often the end users employed a certain function and where they used it. In this way, it must show precise metrics, such as frequency of use and depth of function nesting, of the dataset it conveys.
3 **Promote a qualitative understanding of the patterns.** Lest these patterns remain abstract, the tool should supplement its observations with concrete instances of relevant formulas from the corpus. Ideally, it should even direct users to the exact cell in the spreadsheets where the formula was used, contextualizing the functions.

Likewise, to bound our scope, we outlined a few goals to specifically avoid accomplishing with this:

!1 **Do not try to directly explain what a function does.** Though the tool tries to foster tries understanding by linking pattern to example, it won't provide a precise description of what a function accomplishes. The tool's user must infer this.

!2 **Do not create new formulas.** This is essentially an exploratory tool, not a generative one. It should not produce any information other than new views of the original data.

!3 **Do not visualize individual formulas.** Though there is room to explore the place of a single function in the group, the tool must design the core visualization around the entire body of data, not the other way around.

B. Design

Given the composition of formulas as functions and their arguments (which could be yet more functions), we need a visualization style that conveys this sense of parent-child relationship. As such, we decided to represent formulas as trees where the branching factor depends on both the number of arguments in a function and the number of possible functions observed as an argument in a function.

Guided by these goals, we faced a number of decision points in our design, which we outline below.

1a *Copied formulas*: Excel allows users to spread a formula over an area, repeating the same task in each cell with minor adjustments. Without checking for this, the analysis may not reveal the functions most commonly used together but rather the formulas most often applied to large areas. To combat this, we converted formulas from their native A1 format to the relative R1C1, in which copied formulas should be identical, and reduced the records to unique R1C1 formulas per sheet.

1b *Optional arguments*: How should the tool handle functions which accept a variable number of arguments? SUM, for example, can have anywhere from 1 to 255, and IF can accept either 2 or 3. [3] Without prior evidence, it's possible that spreadsheet programmers use different techniques for different numbers arguments. To account for this, we separate and analyze the different quantities of arguments observed for each function; the tool, however, uses as default all of the options collapsed into a single representation.

2a *Importance of depth*: When a function appears within another, should it be analyzed only as a nested function or would it also be valid to analyze the nested function on its own? If the former, then information about the same function will be scattered across different trees with no way to aggregate them. If the latter, then the tool will analyze some functions multiple times to capture every possible level of nesting. Both approaches have benefits and drawbacks, and so we included both.

2b *Pattern density*: How should the tool quantitatively order its elements: by a function's raw frequency or by the unity of patterns it leads to? For example, if SUM has for its first argument two possibilities, one which itself contains 1000 unique argument possibilities with 1 occurrence each (high frequency, low pattern density) and another seen with 2 argument possibilities of 100 occurrences each (low frequency, high pattern density), which would be more interesting to emphasize? The answer depends on the nuances of the questions, but for simplicity, I've shown the former.

2c *Non-functions*: How should the tool represent everything in the formula that isn't a function: numbers, string literals, errors, references, etc? Since these don't accept arguments, they will adorn the tree as leaves, and their precise content won't affect the functions around them as long as their types are known. As such, in the visualization, all of these nodes are replaced and aggregated under their types.

3a *Suitable examples*: Goal 3 supports the inclusion of examples in the visualization as a way of tying pattern to example, but how should examples be chosen? For this, the simpler is the better, and we made the broad working assumption that shorter (by character length) functions are simpler, and thus chose the shortest available.

3b *Spreadsheet connections*: Is it possible to contextualize these examples even better? Yes, by leading the user directly back to the originating spreadsheet. The tool hyperlinks each example, then, back to the file that contains it, providing sheet, row, and column numbers if it doesn't open directly there.

C. Implementation

We can view the final visualization as the product of two discrete processes:

- *Collection*: Given a set of Excel sheets, the tool, written mostly in Java, uses Apache POI[4] to identify and iterate over every cell containing a valid formula. Afterward, it calls POI's formula parser to break the formula text into an ordered set of individual tokens, which the tool then parses into the tree-like form by which it is recorded. When all formulas have been analyzed like this, it produces JSON files for each top-level function in the set.
- *Presentation*: The JSON files, meanwhile, feed into the presentation code, implemented in Javascript with much help from the visualization library D3[5].

When the presentation code displays the tree, it shows two types of nodes with different meanings: the circles represent a function that has been observed at that structural place, and hovering over these yields a tooltip with supplementary information and examples; and squares, which represent the argument positions for the function which is their parent. For example, if the IF function has been observed with three

[3]https://support.office.com/en-us/article/Excel-functions-by-category-5f91f4e9-7b42-46d2-9bd1-63f26a86c0eb

[4]https://poi.apache.org/

[5]https://d3js.org/

arguments, 3 squares will extend from it as children. Clicking on the square labeled '1' will yield more circles, all of which are functions or values that the tool observed as the first argument in an IF function; clicking on '2' will yield those possibilities in the second position; and so on.

D. Limitations

Because the data collection depends on POI's formula parser, it affords no leeway or partial information from a formula; it either processes it perfectly or throws it out. As such, the visualization inhibits insight into anything with syntactical errors. Note, however, that this does not include standard Excel errors like #REF! and #DIV/0, which the parser handles well.

Unfortunately, the parse also failed to handle any externally defined functions, even if they were syntactically correct in their original context. On one hand, we don't consider the loss of these specific functions to be tragic, as they are not part of Excel proper and thus provide no insight into how people use built-in functions. However, information about other functions within them are lost.

Many functions have an expected order of arguments, but some, like SUM and MATCH, can be reordered in various ways and maintain their value. However, the representation that this tool uses reinforces for all functions that the order is important, even in these exceptions.

IV. Case Study

—

V. Conclusion

—

Acknowledgment

This material is based upon work supported in whole or in part with funding from the Laboratory for Analytic Sciences (LAS). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the LAS and/or any agency or entity of the United States Government.

References

[1] H. Kopka and P. W. Daly, *A Guide to $\LaTeX$*, 3rd ed. Harlow, England: Addison-Wesley, 1999.