Toward Sharing Reasoning to Improve Fault Localization in Spreadsheets

Joseph Lawrance, Margaret Burnett, Robin Abraham and Martin Erwig
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, Oregon 97331
{lawrance,burnett,abraharo,erwig}@eecs.oregonstate.edu

Abstract

Although researchers have developed several ways to reason about the location of faults in spreadsheets, no single form of reasoning is without limitations. Multiple types of errors can appear in spreadsheets, and various fault localization techniques differ in the kinds of errors that they are effective in locating. Because end users who debug spreadsheets consistently follow the advice of fault localization systems [9], it is important to ensure that fault localization feedback corresponds as closely as possible to where the faults *actually* appear.

In this paper, we describe an emerging system that attempts to improve fault localization for end-user programmers by sharing the results of the reasoning systems found in WYSIWYT [13, 14] and UCheck [1, 6]. By understanding the strengths and weaknesses of the reasoning found in each system, we expect to identify where different forms of reasoning complement one another, when different forms of reasoning contradict one another, and which heuristics can be used to select the best advice from each system. By using multiple forms of reasoning in conjunction with heuristics to choose among recommendations from each system, we expect to produce unified fault localization feedback whose combination is better than the sum of the parts.

1 Introduction

Spreadsheet systems like Excel are among the most widely used programming systems. Research estimates that the number of end-user programmers, which includes spreadsheet users, outnumbers professional programmers by an order of magnitude [15]. Both end-user programmers and professional programmers often make mistakes, but end-user programmers rarely possess the organized test suites and knowledge of software engineering methodologies that professional programmers have to mitigate problems. Unfortunately, up to 90% or more of spreadsheets contain faults [7, 10]. Because spreadsheets are often used for important tasks and decisions, faults in them have been tied to costly errors.[1] The potential risks of spreadsheet faults extend beyond monetary costs, particularly in light of the Sarbanes-Oxley Act of 2002, a law which requires corporations to examine the validity of their spreadsheets [8].

Although spreadsheets are essentially a grid of cells, various information bases can be extracted out of spreadsheets, and each information base can highlight different categories of faults. For example, cells often contain explicit relationships to other cells, in the form of cell references, from which data flow graphs emerge; these data flow graphs can be used to identify reference faults[2] [5]. Furthermore, the juxtaposition of row and column headers against cells containing data within spreadsheets typically implies spatial relationships among cells, from which unit inference graphs emerge. Unit inference can be used to identify certain types of reference, range, and omission faults [2]. Other information bases supplied by end users can assist fault localization. For example, the value of cells is often expected to fall within certain intervals; by asserting intervals on cells, cells whose values fall outside their intervals can be located [4, 3, 5]. Adding assertions helped significantly with non-reference faults, suggesting that the addition of assertions into the environment fills a need not met effectively by the data flow testing methodology alone [5]. Furthermore, in several domains, particularly finance, it is often the case that two cells within a spreadsheet must add up to the same value; asserting relationships such as equality among groups of cells can be used to audit spreadsheets. Our work in progress to improve fault localization is based on the assumption that reasoning about faults in only one way is insufficient to locate several different categories of faults effectively.

Our emerging prototype relies on the results of the independent reasoning systems found in UCheck and in WYSIWYT. The two systems base their judgments on different information bases derived from spreadsheets: UCheck analyzes the spatial juxtaposition of row and column headers against data cells, whereas WYSIWYT uses data flow relationships in conjunction with users' judgments to locate faults. By leveraging the reasoning produced from two different information bases, we expect to produce better feedback. We believe that sharing the results of reasoning systems in a way sufficient to locate several categories of faults requires a shared reasoning database and heuristics to resolve competing and sometimes conflicting suggestions from different systems.

[1] http://www.eusprig.org/stories.htm

[2] One classification scheme we have found to be useful in our previous research involves two fault types: reference faults, which are faults of incorrect or missing references, and non-reference faults, which are all other faults.

2 Background

2.1 WYSIWYT (What You See is What You Test)

The fault localization system found in WYSIWYT relies on users checking off at least some of the cell values that are correct (with checkmarks) or incorrect (with X-marks) to locate cells containing faults. By allowing users to incrementally test spreadsheets as they develop them, the WYSIWYT fault localization and testing methodology maintains the interactive nature of spreadsheet systems [12, 11]. WYSIWYT provides automatic, immediate visual feedback about "testedness" for cell values through cell border colors, and users of WYSIWYT are able to improve their test effectiveness without training in testing theory [12]. From users' judgments of cells, WYSIWYT determines fault likelihood for each cell based on the backwards slice of cells marked by users as wrong. WYSIWYT presents fault localization feedback to users by progressively shading cells darker the more likely they contain faults, as shown in Figure 1.

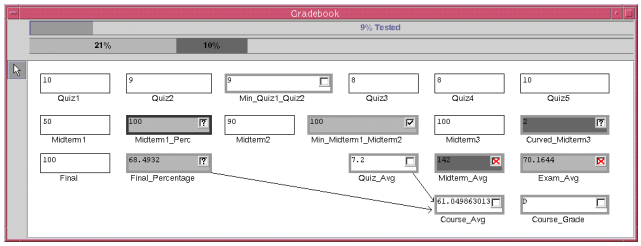

Figure 1. Users' judgments and fault localization feedback

2.2 UCheck

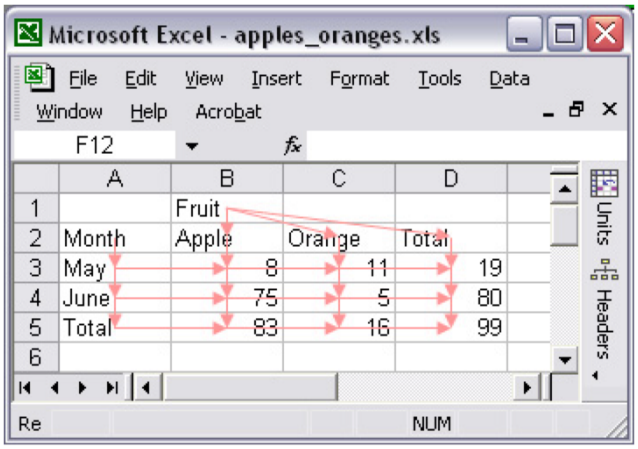

Figure 2. Headers inferred from spreadsheet layout

To locate faults, UCheck first analyses the spatial structure of the spreadsheet to then perform unit inference [1, 6]. UCheck examines the layout to determine the relationship between labels and data cells, as shown in Figure 2. From this information, UCheck can infer the units that apply to all non-blank cells in the spreadsheet. For example, UCheck understands that the unit of cell B3 in Figure 2 represents not just an apple, but also a kind of fruit. UCheck also understands that B3 also is associated with the month of May. From this understanding of units, UCheck can identify when cells inappropriately combine incompatible units, as shown in Figure 3.

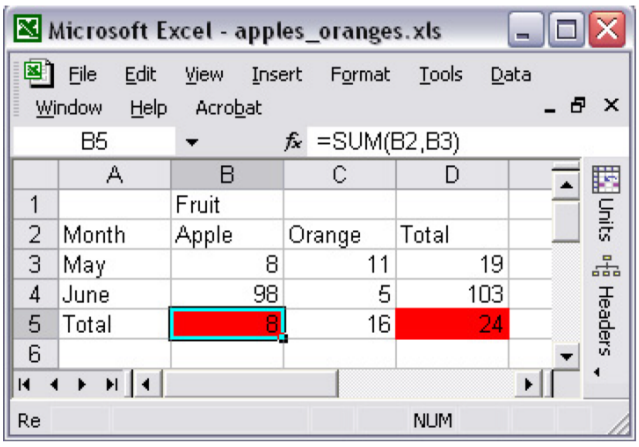

Figure 3. Range error identified from analysis

3 The evaluation testbed

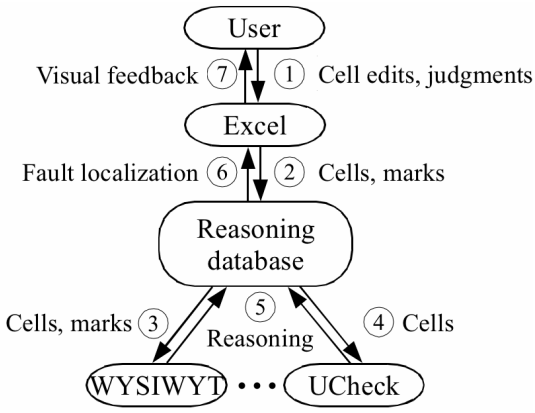

Figure 4. The design of the evaluation testbed

Figure 4 shows the design of the evaluation testbed and the sequential flow of information among the components in the proposed system. Between steps 2 and 6 in Figure 4, the reasoning database propagates cell edits to WYSIWYT and UCheck, then aggregates fault localization reasoning from each system and finally applies heuristics to select and combine fault localization feedback from the two systems to send back to Excel. Note that the design depicted in Figure 4 suggests the possibility of including additional reasoning systems in the future; for now, only the feedback from WYSIWYT and UCheck are used.

Evaluating the proposed system requires a comparison of the known faults in a spreadsheet with the feedback generated by WYSIWYT, UCheck, and the combined feedback from the two systems. Table 1 shows the four possible ways fault localization feedback corresponds to the actual faults for each cell.

Table 1. Fault localization feedback vs. actual faults

	Cell formula	
Fault localization feedback	Right formula	Faulty formula
Cell is Correct	CR	CF
Cell is Incorrrect	IR	IF

We are in the process of implementing our prototype so as to empirically investigate the following questions:

- How well do these systems compare in correctly locating faults (IF)?
- When do these systems falsely identify correct cells as faults (IR)?
- When do these systems falsely identify faulty cells as correct (CF)?
- When do the systems diasgree in their feedback?
- What heuristics are most effective in selecting and combining feedback?

4 Conclusion

We have presented our work in progress on experimenting with and empirically evaluating the effectiveness of sharing the results from multiple reasoning systems to improve spreadsheet fault localization. We hope that this approach will prove flexible and beneficial enough to allow a large portfolio of reasoning devices to be brought to bear on spreadsheet errors.

5 References

[1] R. Abraham and M. Erwig. Header and unit inference for spreadsheets through spatial analyses. In *IEEE Symp. on Visual Languages and Human-Centric Computing*, pages 165–172, 2004.

[2] R. Abraham and M. Erwig. How to communicate unit error messages in spreadsheets. In *WEUSE I: Proceedings of the first workshop on End-user software engineering*, pages 1–5, New York, NY, USA, 2005. ACM Press.

[3] Y. Ayalew. *Spreadsheet Testing Using Interval Analysis*. PhD thesis, Universität Klagenfurt, 2001.

[4] Y. Ayalew, M. Clermont, and R. Mittermeir. Detecting errors in spreadsheets. In *Proceedings of EuSpRIG 2000 Symposium: Spreadsheet Risks, Audit and Development Methods*, 2000.

[5] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace. End-user software engineering with assertions in the spreadsheet paradigm. In *International Conference on Software Engineering*, pages 93–103, 2003.

[6] M. Erwig and M. Burnett. Adding apples and oranges. In *4th Int. Symp. on Practical Aspects of Declarative Languages*, pages 173–191, 2002.

[7] R. R. Panko. Spreadsheet Errors: What We Know. What We Think We Can Do. In *Proceedings of the Spreadsheet Risk Symposium, European Spreadsheet Risks Interest Group (EuSpRIG)*, 2000.

[8] R. R. Panko and N. Ordway. Sarbanes-Oxley: What about all the spreadsheets? In *European Spreadsheet Research Information Group*, 2005.

[9] A. Phalgune, C. Kissinger, M. Burnett, C. Cook, L. Beckwith, and J. R. Ruthruff. Garbage in, garbage out? An empirical look at oracle mistakes by end-user programmers. In *IEEE Symposium on Visual Languages and Human-Centric Computing*, 2005.

[10] K. Rajalingham, D. R. Chadwick, and B. Knight. Classification of spreadsheet errors. In *Symp. of the European Spreadsheet Risks Interest Group (EuSpRIG)*, 2001.

[11] G. Rothermel, M. Burnett, L. Li, C. Dupuis, and A. Sheretov. A Methodology for Testing Spreadsheets. *ACM Trans. Software Engineering and Methodology*, 10(1):110–147, 2001.

[12] K. J. Rothermel, C. R. Cook, M. M. Burnett, J. Schonfeld, T. R. G. Green, and G. Rothermel. WYSIWYT testing in the spreadsheet paradigm: An empirical evaluation. In *ICSE '00: 22nd International Conf. Software Engineering*, pages 230–239, 2000.

[13] J. Ruthruff, E. Creswick, M. Burnett, C. Cook, S. Prabhakararao, M. F. II, and M. Main. End-user software visualizations for fault localization. In *Proceedings of ACM Symposium on Software Visualization*, pages 123–132, 2003.

[14] J. R. Ruthruff, S. Prabhakararao, J. Reichwein, C. Cook, E. Creswick, and M. Burnett. Interactive, visual fault localization support for end-user programmers. *Journal of Visual Languages and Computing*, 16(1-2):3–40, 2005.

[15] C. Scaffidi, M. Shaw, and B. Myers. Estimating the numbers of end users and end user programmers. In M. Erwig and A. Schürr, editors, *IEEE Symposium on Visual Languages and Human Centric Computing*, pages 207–214, 2005.