Identifying Structure in Spreadsheets*

Robin Abraham
Oregon State University
abraharo@cs.orst.edu

Abstract

Spreadsheets are among the most commonly used programming systems in the world. Existing spreadsheets are rife with errors, some of which have serious impacts. We are working on algorithms and strategies that automatically infer the structure of spreadsheets. This information can be used to develop systems that enable end users to develop safer spreadsheets.

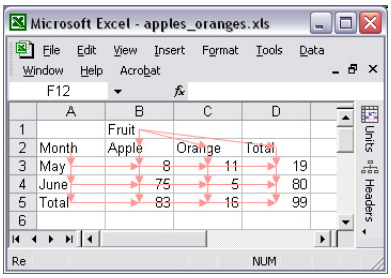

Figure 1. Automatically inferred headers.

1. Introduction

Every year, hundreds of millions of spreadsheets are being created by tens of millions of professionals and managers [6]. Studies have shown that spreadsheets have high error rates [6, 8]. The spreadsheet programmers on the other hand, are unaware of this aspect and tend to have a high level of confidence in the correctness of their spreadsheets [6]. Both of these factors combine to produce the expensive failures reported in literature.

Despite the fact that they are error-prone, spreadsheets are also used in Science and Mathematics education [5] since spreadsheet systems are powerful and relatively easy to use. They can serve as vehicles for the introduction of computers and elementary modeling and programming concepts to students. With widespread adoption of spreadsheets, there is a greater need for better auditing tools. Even so, spreadsheet auditing requires too much time, effort, and expertise [6]. Panko has shown that code inspection or auditing is difficult. Only half or fewer errors can be detected using these approaches and the costs are already very high for medium size spreadsheets [7]. Moreover, it would be unreasonable to expect users disadvantaged by their background, education, or learning style to carry out effective code inspections and spreadsheet testing since they do not have any prior experience in design, development, and maintenance of software systems. Unfortunately, the end users also do not have too many tools at their disposal (or the training) to detect and correct errors in their spreadsheets. In this context, tools that perform reliable and automatic checking of the spreadsheet are of paramount importance.

*This work is supported by the National Science Foundation under the grant ITR-0325273 and by the EUSES Consortium.

2. Spreadsheet Structure Identification

In previous work, we have investigated two approaches to help end users build error-free spreadsheets. The Gencel system [2] adopts a top-down approach to developing spreadsheets. This approach requires the user to build the spreadsheet specification first and then use the specification to build/generate the spreadsheet instances. The top-down approach might give rise to the problems of premature commitment and viscosity [4] if it is not easy to make changes to the model level once the spreadsheet instance has been created. The unit system [3] on the other hand, adopts a bottom-up approach. The user can invoke the unit system while working with any arbitrary spreadsheet and it will first perform automatic header inference using spatial analysis algorithms [1] to infer the units and then perform unit checking and report the detected errors to the user.

Figure 1 shows the headers that have been automatically inferred in a harvest spreadsheet. The arrows point from header cells to target cells. In the spreadsheet shown in Figure 2, the user has accidentally entered a reference to cell C3 in cell B4. This reference is obviously an error since it implies that the cell B4 contains the number of apples harvested in June (by virtue of its position with respect to the headers) *and* the number of oranges harvested in May (by virtue of the reference). Excel does not detect this error. When the user clicks the *Units* button on the toolbar on the

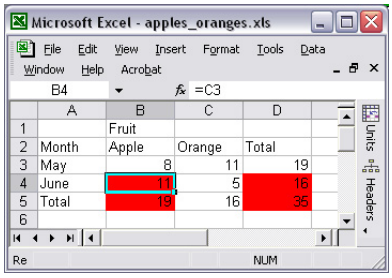

Figure 2. Spreadsheet with unit errors.

right, the unit system marks cell B4 as the site of a unit error. Cells B5, D4, and D5 have SUM formulas that reference B4. Therefore, they too, are marked as sites of unit errors. The unit inference is based on the header information derived by spatial analysis of the spreadsheet.

Even though end users might perform spreadsheet programming as more of an exploratory activity, the final result is a mapping of the user's conceptual model onto a two-dimensional grid interface. The conceptual model determines the structures within the spreadsheet. Since the systems do not require the user to explicitly provide this information, reconstruction of the model by reverse engineering a given spreadsheet is a non-trivial task. The possibility of a wide variety of models mapped onto the semantically unconstrained grid structure introduces a high level of ambiguity into the task of model inference.

We are currently investigating the problem of accurately identifying structures within spreadsheets since awareness of the structure would allow the system to be of help to the user in many ways. Three examples are presented below.

1. **Accurate and automatic spreadsheet error-detection**
   As mentioned earlier in this section, automatic header inference allows us to automate the entire process of unit checking. Along similar lines, the development of more robust techniques for inference of structures in spreadsheets would go a long way in making automatic and reliable spreadsheet audit possible, thereby addressing the concerns of cost and effort expressed in [6, 7].

2. **Enforcement of best practices**
   Beginning students are given some introduction to *safe* programming practices during introductory courses on spreadsheets. It might not even be feasible for the students to remember and conform to these recommendations if they are in the form of long lists of *do*'s and *don't*'s. It would be useful in such a situation to have a *spreadsheet listener* running in the background, constantly keeping track of the user actions and inferring changes to the spreadsheet model. The listener could then guide the students so that they conform to the best practices.

3. **Spreadsheet structure parsing**
   Legacy spreadsheets are a huge barrier to the adoption of any new system like Gencel [2]. Companies and individual users might have complex (and potentially very large) spreadsheets whose development has consumed a considerable amount of time and money. They might not be willing to port these spreadsheets to some new system (in spite of exciting advantages in terms of new features) either because of the effort involved or because the model behind the original spreadsheet is not clear anymore. Spreadsheet structure parsing would be the only solution in such situations—to either convert the spreadsheet to a totally different representation or to extract the specification (to be used in a structured spreadsheet system).

3. Conclusions

The increase in usage coupled with high incidence of errors in end-user spreadsheets warrants investigation into techniques and frameworks for the development of error-free spreadsheets. To support users who are disadvantaged by their background, education, learning style, or physical characteristics, it is important that such systems only require minimal user intervention to ensure correctness of the spreadsheet. In this context, the development of algorithms that are accurate enough to carry out automatic identification of structures in spreadsheets is of vital significance. This capability would enable us to provide the user with better tools for developing reliable spreadsheets.

References

[1] R. Abraham and M. Erwig. Header and Unit Inference for Spreadsheets Through Spatial Analyses. *IEEE Symposium on Visual Languages and Human-Centric Computing*, 2004.

[2] M. Erwig, R. Abraham, and S. Kollmansberger. Gencel—A Program Generator for Correct Spreadsheets. *Submitted for publication*, 2004.

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

[4] T. R. G. Green and M. Petre. Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. *Journal of Visual Languages and Computing*, 7(2):131–174, 1996.

[5] E. Neuwirth. Spreadsheets, mathematics, science and statistics education. `http://sunsite.univie.ac.at/Spreadsite/`.

[6] R. R. Panko. What we know about spreadsheet errors. *Journal of End User Computing (Special issue on Scaling Up End User Development)*, 10(2):15–21, 1998.

[7] R. R. Panko. Applying code inspection to spreadsheet testing. *Journal of Management Information Systems*, 16(2):159–176, Fall 1999.

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