Using Calculation Fragments for Spreadsheet Testing and Debugging

Dietmar Jannach, Thomas Schmitz

Department of Computer Science, TU Dortmund, Germany

dietmar.jannach@tu-dortmund.de, thomas.schmitz@tu-dortmund.de

Abstract—**A number of automated techniques and tools were proposed in the research literature over the years which aim to support the spreadsheet developer in the process of testing and debugging a faulty spreadsheet. One underlying assumption of many of these approaches is that the spreadsheet developer is capable of providing test cases or is at least reliably able to determine whether a calculated value in a certain cell is correct given the current set of inputs.**

Since real-world spreadsheets can be complex, we argue that these assumptions might be too strong in some situations. We therefore propose to support the user during testing and debugging by automatically computing *spreadsheet fragments* of manageable size. The spreadsheet developer can then verify the correctness of a smaller set of formulas for which the calculated output can be more easily validated.

I. INTRODUCTION

In environments like MS Excel, the set of built-in features designed to support the users when testing or debugging their spreadsheet programs is quite limited. Examples of the few available features include the visualization of individual cell dependencies or the markup of cells containing suspicious formulas ("smells" in the sense of [3] or [5]). Over the last decades a variety of different tools and techniques were proposed in the research literature to help the user avoid, locate and remove errors in spreadsheets [8]. The proposed approaches range from intelligent visualizations over test case generation to the application of novel debugging techniques.

Many of these testing and debugging approaches assume that the users are able to provide test cases, which contain the expected values for the output cells of their spreadsheets [2], [7]; some techniques require the user at least to reliably indicate if an output cell value or a test case is correct or faulty [1], [6]. Providing expected values or even assessing the correctness of individual values might, however, be challenging for the user in particular when the spreadsheets are large and when no known-to-be-correct test cases are available.

We therefore propose to (automatically) modularize the spreadsheet under investigation and ask the user for feedback on the correctness of calculations of smaller and more comprehensible spreadsheet *fragments*. These fragments in some sense correspond to the concept of *unit tests* in standard software development processes based on which the correctness of smaller functional parts of the program can be validated.

The proposed fragment extraction approach is also related to automated refactoring techniques for long methods in imperative programs based, e.g., on control flow and data flow graphs like in [9]. However, spreadsheets have no standard control flow graphs and the data flow graphs are much simpler, because the formulas in the cells do not directly change the values of other cells. Existing refactoring approaches therefore cannot be directly applied and further investigations are required to assess how these methods can be adapted to be applicable for the special and defined structure of spreadsheets.

II. EXAMPLE

Consider the example in Figure 1, which shows the dependency structure of a typical financial calculation sheet adapted from [7]. In the upper part of the spreadsheet (Fragment A), monthly sales data are aggregated with the help of a number of copy-equivalent rows. In the lower part (Fragment B), further data aggregation is done and additional calculations are made on the aggregate values. Structurally or semantically different cells are denoted by differently styled circles, e.g., the empty circles denote input cells or fixed parameters.
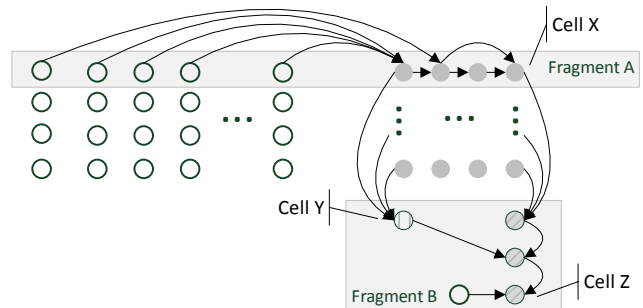

Fig. 1. A typical spreadsheet with fragments.

Let us assume that the spreadsheet developer observes that the "final outcome" in the lower right part of the spreadsheet (Cell Z) only meets his expectations for some but not all tested input value constellations. However, providing the exact expected values – as required by some automated debugging methods [2], [7] – might be cumbersome for the user and error-prone in particular when the spreadsheet is complex.

In this example, we could – as indicated in Figure 1 – have two fragments that can be tested individually and in the case of Fragment A use one row of the spreadsheet as a representative for the other copy-equivalent rows. These fragments could be defined manually by the user; in our work, we are, however, interested in techniques to automatically identify possible fragments and provide adequate tool support.

III. Design Considerations - Technical Approach

A. End User Perspective

As spreadsheet developers are usually not IT experts or programmers, special care has to be taken when designing a tool for fragment-based testing and debugging. We therefore plan to integrate the sketched techniques in our model-based debugging plug-in to Excel called EXQUISITE [7]. When using this tool, the users can stay within their usual spreadsheet environment; appropriate user-oriented metaphors for those concepts that are not part of typical spreadsheet tools (like test cases, fragments, or unit tests), however, still have to be found and evaluated with users.

Another open question in that context is related to the optimal complexity of the fragments a user should work on. If the fragments are too small, test cases for too many fragments have to be defined by the user; if they are complex and span major parts of the spreadsheet, the cognitive effort for the user when providing the test cases might be too high.

B. Automated Extraction of Fragments

The main goal of our work is to develop algorithms and heuristics to automatically extract potentially overlapping fragments that are manageable in size and in the best case semantically connected. These fragments can then for example serve as a basis for the manual or automated creation of test cases, e.g., using property-based testing techniques [4], where the goal is to automatically create test cases which falsify user-specified properties of the output values. In another scenario, the fragments can be used to inspect parts of the spreadsheet which are considered to be "suspicious" as a result of a diagnosis or smell detection technique. We are currently exploring the following fragment construction strategies.

1) Collapsing Copy-Equivalent Structures: In Figure 1, only the last four cells on the right of Fragment A (including Cell X) contain formulas. When the right-most cells of each row are considered suspicious, it can be sufficient to create a fragment which comprises *one representative row*, so that the user only has to create one test case for all rows.

On the other hand, the left-most cell of Fragment B (Cell Y) is based on the values of several copy-equivalent cells and, e.g., corresponds to the sum of the monthly sales figures. If we defined this cell and a subset of its inputs as another Fragment C (not shown in Figure 1), the fragment could contain two or more of the copy-equivalent rows such that the user can validate the correctness of the aggregation function, which would be impossible when there is only one input row. With this second technique the user could, however, miss range errors such as an omitted cell of a sum.

2) Limiting the Dependency Paths: Fragments of manageable size like Fragment B in Figure 1 can be obtained by limiting the depth and maybe also the breadth of the dependency paths leading to a suspicious cell. The depth limitation could be based on simple path length restriction heuristics or based on structural or semantic considerations, e.g., by observing that we encounter a set of copy-equivalent cells.

Existing techniques developed for spreadsheet visualization and comprehension could be applicable in that context.

C. Interactive Testing and Debugging

The provision of adequate tool support when testing or debugging a calculation fragment is finally a crucial part of our approach. From a UI perspective, one could for example create a new worksheet which only contains the cells and formulas of the fragment under investigation. This could however remove context information, as neighboring cells outside the fragment would not be visible. Therefore, we propose to only visually dim the cells outside the fragment and make them read-only to help the user focus on the current task.

For debugging purposes, our model-based diagnosis approach presented in [7] can be used to find the possible causes whenever there is a discrepancy between the expected and the calculated value of an output cell of a fragment. Because of the small sizes of the fragments, we conjecture that the combination of these approaches could help to quickly find the faulty formulas.

In addition, the proposed test and debugging environment should be able to automatically create appropriate test cases which are easy to validate manually. Again, existing approaches for test case generation from the literature should be applied. Note that when creating these test cases, input values have to be generated only for the cells at the fragment borders (e.g., Cell Y in Fragment B). The developed software environment and Excel plug-in has furthermore to be extended in a way that test cases can be easily stored, adapted and automatically executed, e.g., for regression testing.

IV. Summary

We argue that providing exact values for expected calculation outcomes in spreadsheet testing and debugging scenarios might be too challenging for users in some situations. We therefore propose to further investigate approaches in which the given spreadsheet is decomposed into smaller fragments which can be more easily validated by the user.

References

[1] R. Abraham and M. Erwig. AutoTest: A Tool for Automatic Test Case Generation in Spreadsheets. In *Proc. VL/HCC*, pages 43–50, 2006.

[2] R. Abraham and M. Erwig. GoalDebug: A Spreadsheet Debugger for End Users. In *Proc. ICSE*, pages 251–260, 2007.

[3] R. Abreu, J. Cunha, J. P. Fernandes, P. Martins, A. Perez, and J. Saraiva. Smelling faults in spreadsheets. In *Proc. ICSME*, pages 111–120, 2014.

[4] G. Fink and M. Bishop. Property-based testing: A new approach to testing for assurance. *SIGSOFT Softw. Eng. Notes*, 22(4):74–80, 1997.

[5] F. Hermans, M. Pinzger, and A. van Deursen. Detecting and Visualizing Inter-Worksheet Smells in Spreadsheets. In *Proc. ICSE*, pages 441–451, 2012.

[6] B. Hofer, A. Riboira, F. Wotawa, R. Abreu, and E. Getzner. On the Empirical Evaluation of Fault Localization Techniques for Spreadsheets. In *Proc. FASE*, pages 68–82, 2013.

[7] D. Jannach and T. Schmitz. Model-based diagnosis of spreadsheet programs – A constraint-based debugging approach. *Autom. Softw. Eng.*, online, Feb. 2014.

[8] D. Jannach, T. Schmitz, B. Hofer, and F. Wotawa. Avoiding, finding and fixing spreadsheet errors - a survey of automated approaches for spreadsheet QA. *Journal of Systems and Software*, 94:129–150, 2014.

[9] K. Maruyama. Automated Method-Extraction Refactoring by Using Block-Based Slicing. In *Proc. SSR*, pages 31–40, 2001.