A Spreadsheet Cell-Meaning Model for Testing

Daniel Kulesz
Institute of Software Technology
University of Stuttgart
daniel.kulesz@informatik.uni-stuttgart.de

ABSTRACT

Most attempts to test spreadsheets use the spreadsheet-internal model to automatically detect input, intermediate and output cells. In this paper, we discuss an example which shows why this approach is problematic even for simple spreadsheets. Following this, we derive a number of requirements for more feasible spreadsheet cell-meaning models and describe a first prototype we have designed.

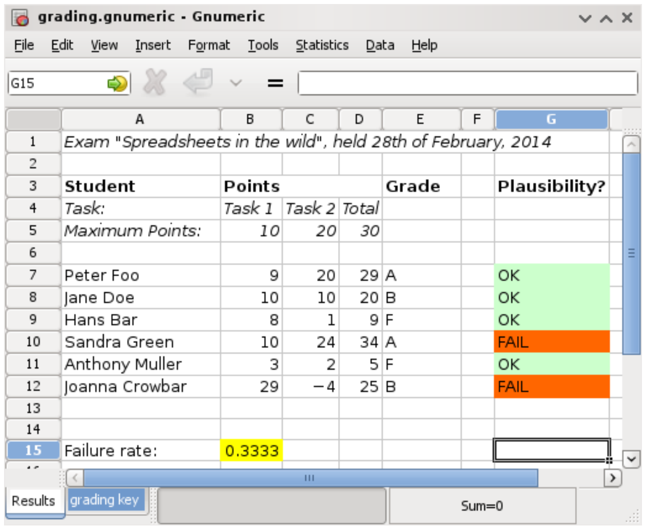
1. INTRODUCTION

Regardless of the hazards which can arise when using spreadsheets, most businesses today regard them as indispensible tools for supporting their processes. This fact, together with the billions of existing spreadsheets [7], indicates a strong need for finding faults in spreadsheets.

There are many approaches for finding faults in spreadsheets [6]. One of them is testing, where the spreadsheet's input cells are populated with values and the values in the spreadsheet's output cells checked for certain criteria (e.g. [5, 3]). To accomplish this, knowing which of the spreadsheet's cells are input cells and which are output cells is mandatory. However, virtually no testing approaches use a designated model for this purpose. Instead, they rely on the model that spreadsheet execution environments use internally for (re)calculation purposes, and which can easily be extracted when considering the dependencies between cells:

- Input cells: Non-formula cells referenced by formula cells
- Intermediate cells: Formula cells referenced by other cells and referencing other cells themselves
- Result cells: Formula cells not referenced by other cells but referencing other cells themselves

We will refer to this model as the 'naive model' throughout this paper. While this model is certainly correct from a technical point of view, looking at it from the perspective of a spreadsheet user's domain can lead to imminent conflicts in a number of cases. In this paper we discuss these cases and propose an explicit model which is more difficult to extract but which we believe is better suited for testing spreadsheets.


grading.gnumeric - Gnumeric
File Edit View Insert Format Tools Statistics Data Help
G15
Results
grading key
Sum=0

	A	B	C	D	E	F	G
1	*Exam "Spreadsheets in the wild", held 28th of February, 2014*						
2							
3	**Student**	**Points**			**Grade**		**Plausibility?**
4	*Task:*	*Task 1*	*Task 2*	*Total*			
5	*Maximum Points:*	*10*	*20*	*30*			
6							
7	Peter Foo	9	20	29	A		OK
8	Jane Doe	10	10	20	B		OK
9	Hans Bar	8	1	9	F		OK
10	Sandra Green	10	24	34	A		FAIL
11	Anthony Muller	3	2	5	F		OK
12	Joanna Crowbar	29	−4	25	B		FAIL
13							
14							
15	Failure rate:	0.3333					

Figure 1: Example for a grading spreadsheet

Since the model is concerned with the type of cells from the perspective of what the cells mean to users, we could refer to it as "cell-type model". Unfortunately, the term "cell-type" is usually already used for describing the data type of a spreadsheet cell's contents. To avoid confusion, we use the notion of "cell-meaning model" instead.

2. ISSUES WITH THE NAIVE MODEL

Figure 1 shows a spreadsheet for managing grades of a study course. The spreadsheet is filled with data by a course instructor and later passed to a secretary for transferring the grades to a different system. Furthermore, the spreadsheet is used for statistical purposes by the manager of the study course this exam belongs to. We want to use this spreadsheet as a showcase with counter examples, arguing why the naive model can lead to a biased perception of the actual input cells and output cells of a spreadsheet:

- The total points (D7 to D12) could be output values for a secretary who has to process these grades further (e.g. write letters to students), but these cells are referenced by the grade cells (E7 to E12). Thus, the cells would be regarded as intermediate cells by naive models and not as output cells.

- The study program manager might not be interested at all in the total points of the particular students but only in the failure rate (B15). Thus, he would not see the grades as output cells.
- Spreadsheet authors sometimes use defensive programming techniques and introduce checks themselves. The plausibility column (G) is such an example: It checks whether any of the grades for Task 1 or Task 2 are outside acceptable limits (zero to maximum points for the task). The naive model would treat the cells in the plausibility column (G7 to G12) as output as well.
- The second worksheet named ‘grading key‘ is referenced by VLOOKUP functions in the grade column's cells (E7 to E12). The naive model would therefore interpret these referenced cells as input cells. However, none of the users of this spreadsheet is supposed to change the contents of these cells as they contain merely static data.

3. REQUIREMENTS

We are convinced that biased perceptions can be reduced if a spreadsheet is tested using a model explicitly designed for this purpose. From the discussion in the previous section, we derive the following requirements for such a model:

- **User-specifiable:** It must be possible for users to specify the cells themselves. If automatic extraction is used, users must be able to change it.
- **Support for views:** Since users have different perceptions and needs of the same spreadsheet, we either need one testing model with different views or it must be allowed that more than one testing model instance per spreadsheet exists.
- **Input cell types:** The model must separate input cells at least into two types: data cells (which contain data that rarely changes or is fed from another system) and actual decision variables which are supposed to be manipulated by the user (of this model).
- **Output cell types:** Apart from intermediate cells, the model must support cells which provide the data with final results (which the user is looking for) as well as support for plausibility and other additional cells.

Apart from these rather theoretical requirements, we identified two major practical requirements for the success of the implementation of such a model:

- **Understandability:** It must be easy for spreadsheet users to understand the model with no or little training, so that users can identify cells correctly in the sense of the model.
- **Acceptance:** Even if the model would be easily understandable for spreadsheet users, its benefits must be striking so that spreadsheet users will be motivated to take the effort connected with using the model. (Basically this requires a proper attention investment model as described by Blackwell and Burnett [2] [1]).

It seems infeasible to expect spreadsheet users to identify all cell-meanings manually, especially for huge spreadsheets. As already discussed, fully automated cell-meaning detection seems impossible — but it might be beneficial to consider assisting users by *proposing* cell-meanings based on auto-detection techniques.

A promising starting point could be the work of Hermans [4] which tries to identify plausibility cells automatically by inspecting result cells for two additional constraints: the formula starting with the IF operation and containing at least one branch which can result in a string. While this certainly works in many cases (including our small example), we have already seen spreadsheets which use numeric outputs for plausibility cells so this approach would fail. Yet, since such cases are pretty rare, asking users to just validate auto-detected cell-meanings instead of asking them to specify cell-meanings themselves might result in lower overall effort and thus higher acceptance.

4. PROTOTYPE

We prototyped a cell-meaning model which takes into account the requirements stated in the previous section. The model is illustrated in Figure 2 as a standard UML class diagram.

We provide a partial implementation of our prototype in our tool ‘Spreadsheet Inspection Framework‘ which is available as open source software from GitHub[1]. The implementation allows users to manually mark cells and use them later for specifying test cases, but does not support all proposed cell-meaning types yet and lacks auto-detection capabilities.

5. FUTURE WORK

To assess the feasibility of the model, further research is required. A crucial aspect for evaluation will be the question whether the cell types can be communicated clearly to users, so users will tag existing spreadsheet cells according to our proposed model.

Another important aspect will be the acceptance of the model. Since we believe that acceptance might be very low without a reasonable level of automated assistance, it seems worthwhile to address this point first.

Although we explained the theoretical limitations of the naive model in this work, it must be explored whether the additional manual effort connected with applying our model yields enough benefits in terms of its ability to detect faults in spreadsheets more accurately.

[1]https://github.com/kuleszdl/Spreadsheet-Inspection-Framework

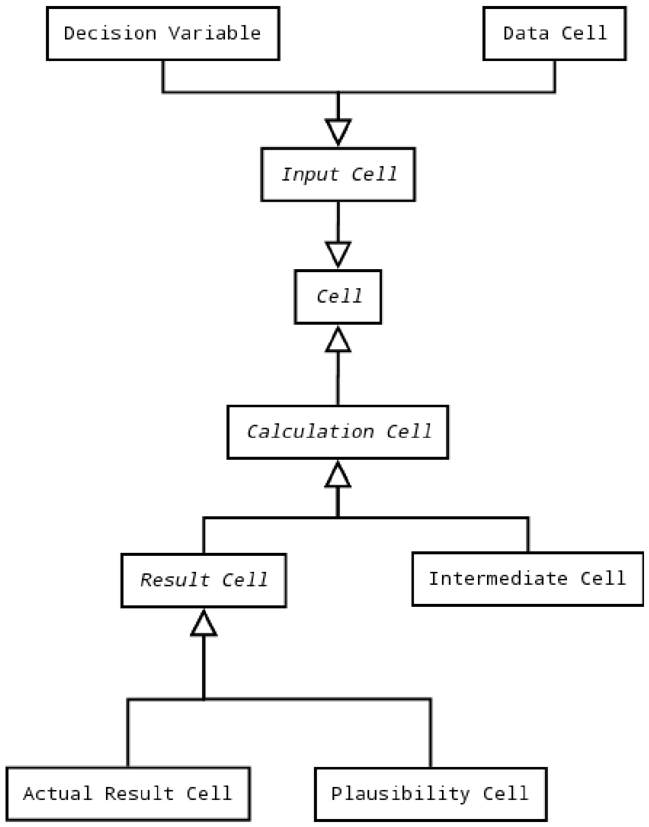


Figure 2: The cell-meaning model we propose

6. ACKNOWLEDGEMENT

We would like to thank Zahra Karimi, Kornelia Kuhle, Mandy Northover, Jochen Ludewig and Stefan Wagner for their constructive feedback on earlier versions of this position paper. Furthermore, we received many good hints and comments from the reviewers for which we are very thankful.

7. REFERENCES

[1] A. Blackwell and M. Burnett. Applying attention investment to end-user programming. In *Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on*, pages 28–30. IEEE, 2002.

[2] A. F. Blackwell. First steps in programming: A rationale for attention investment models. In *Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on*, pages 2–10. IEEE, 2002.

[3] M. Fisher II, G. Rothermel, D. Brown, M. Cao, C. Cook, and M. Burnett. Integrating automated test generation into the wysiwyt spreadsheet testing methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 15(2):150–194, 2006.

[4] F. Hermans. Improving spreadsheet test practices. *Center for Advanced Studies on Collaborative Research, CASCON*, 2013.

[5] D. Jannach, A. Baharloo, and D. Williamson. Toward an integrated framework for declarative and interactive spreadsheet debugging. In *Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pages 117–124. SciTePress, 2013.

[6] 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*, 2014.

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