Exploiting Domain-Specific Structures For End-User Programming Support Tools* — Position Paper —

Robin Abraham Martin Erwig
Oregon State University

1. PROGRAMMING LANGUAGE RESEARCH FOR SPREADSHEETS

In previous work we have tried to transfer ideas that have been successful in general-purpose programming languages and mainstream software engineering into the realm of spreadsheets, which is one important example of an end-user programming environment. More specifically, we have addressed the questions of how to employ the concepts of type checking, program generation and maintenance, and testing in spreadsheets. While the primary objective of our work has been to offer improvements for end-user productivity, we have tried to follow two particular principles to guide our research.

(1) Keep the number of new concepts to be learned by end users at a minimum.
(2) Exploit as much as possible information offered by the internal structure of spreadsheets.

In the following we will illustrate our research approach with several examples.

The idea behind the UCheck system [8] is to interpret the labels in a spreadsheet as annotations akin to type declarations in traditional programs. By identifying rules that express how labeled cells can be combined in formulas in a meaningful way, the information about cell labels can then be exploited to check the consistency of spreadsheet formulas [13]. To make this approach feasible we needed a way to automatically infer the information about which labels are to be used as type information and which cells are annotated by which labels [1], because a tool that required a spreadsheet user to annotate a spreadsheet with this information would probably not be very widely used due to the high additional cost involved. We have also begun to investigate ways to infer from the inconsistent use of labels in formulas suggestions for changes in formulas that can be reported to the end user [3].

We have also investigated a different approach to type checking that is based on the traditional notion of types, extended by a concept of formula shapes [6]. In addition to finding errors in spreadsheets, this approach can also be used to infer spreadsheet models, an aspect to be discussed below.

A strong point about the type checking approaches is that they operate fully automatically—all an end user has to do is to click a button, and sources of potential errors are found and highlighted instantly. At the same time, this advantage can also mean a drawback since end users might rely too much on the system, in particular, they might assume that their spreadsheet is correct when UCheck does not report an error, not knowing or ignoring the fact that automatic type checking cannot be complete in the sense of finding all errors in a program. Therefore, other methods to finding errors are needed to complement automatic type checking. One example is the WYSIWYT approach, invented by Rothermel, Burnett, and others [15], which supports end users with systematically testing their spreadsheets. An important objective of testing is to achieve sufficient coverage of the program being tested. Supporting the user in finding test cases attaining high coverage is the goal of automatic test case generators.

We have developed one such tool called "AutoTest" [4], which generates test suites that obtain 100% DU-coverage (for reachable code). This is an improvement over a previous approach, "Help Me Test", that was developed for the WYSIWYT framework [14]. AutoTest is also considerably more efficient than Help Me Test.

Once a user has identified through testing that a cell contains a wrong value, the next problem is to find out where the error is located in the spreadsheet and how to correct it. To this end, we have developed a method called "goal-directed debugging", or "GoalDebug", which asks the user for a correct value for that cell and then computes a ranked list of suggested changes for formulas, each of which would cause the specified target value to be computed [2]. These changes can be automatically applied, which eliminates a whole class errors introduced by end users during the editing of formulas. Using a systematic study based on mutation testing, we have found that GoalDebug consistently presents the correct changes among the most highly ranked suggestions [7].

While all the previously mentioned approaches try to detect errors, the goal of the Gencel system [11] is to prevent the introduction of errors into spreadsheets. The system is based on a concept of templates that capture the potential evolution of a spreadsheet over time. Changes to spreadsheets, such as insertion and deletion of (groups of) rows and columns are controlled by these templates that ensure the formulas will always be adjusted correctly. In fact, we can prove that spreadsheets maintained by Gencel based on these templates are always free from type, range, and reference errors [12]. Templates have a visual representation that is almost identical to the notation known to end users

*This work is partially supported by the National Science Foundation under the grant ITR-0325273 and by the EUSES Consortium (`http://EUSESconsortium.org`).

Dagstuhl Seminar Proceedings 07081
End-User Software Engineering
http://drops.dagstuhl.de/opus/volltexte/2007/1086

from spreadsheets [9] and can be created using a visual editor. We have also developed a method to infer templates from existing spreadsheets, which facilitates the use of Gencel for legacy spreadsheets [5]. The templates inferred by our system have been judged by experts to be better than those developed by novice and even expert users. We have extended the Gencel model to include more high-level modeling features while still retaining its visual attractiveness. The resulting ClassSheets model [10] also allows the integration of spreadsheet modeling into the UML modeling process.

All of our approaches to improve the quality of spreadsheets essentially exploit in some way or another the

- *simplicity* of the spreadsheet language, and
- *embedding* of computations in a spatial grid.

These two aspects allow the reasoning to be (a) simple enough because complicated language features, such as recursion and nested scope, need not be addressed and (b) supported by the spatial structure exhibited by the arrangement of cells.

2. FUTURE RESEARCH

Research for general-purpose languages has been quite successful and has produced important results from which most professional programmers benefit today. An example are the sound type systems now to be found in mainstream programming in languages, such as Java.

We have demonstrated with UCheck that it is indeed possible to bring the benefits of tools successfully employed in general-purpose languages to the realm of spreadsheets. Similarly, the Gencel/ClassSheets systems show that the idea of high-level modeling, as known from UML, can be employed successfully in the spreadsheet domain.

These experiences suggests as a successful strategy for future research:

> *Redesign methods known from general-purpose languages for end-user programming domains by exploiting application-specific structures and practices.*

In the spreadsheet domain, the spatial layout of cells entails the practice of end users to place closely related items in the same area, or in the same row or column. It is this combination of spatial structure and corresponding user practice that lets tools like UCheck or GoalDebug work so well. Therefore, identifying and exploiting such links might be a key step in designing successful end-user programming tools.

Example areas for new potential spreadsheet tools to be investigated are refactoring, version control, and use cases, to name just a few.

We believe that the successful transfer of concepts also requires a critical mass of researchers working in that area, which is currently hardly the case. Therefore, a second goal should be the following.

> *Persuade programming language and software engineering researchers to participate in the development of tools for end-user programming.*

The three most relevant papers are the following.

- UCheck, *JVLC 2007* [8]
- Gencel, *JFP 2006* [12]
- GoalDebug, *ICSE 2007* [7]

3. REFERENCES

[1] R. Abraham and M. Erwig. Header and Unit Inference for Spreadsheets Through Spatial Analyses. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*, pages 165–172, 2004.

[2] R. Abraham and M. Erwig. Goal-Directed Debugging of Spreadsheets. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*, pages 37–44, 2005.

[3] R. Abraham and M. Erwig. How to Communicate Unit Error Messages in Spreadsheets. In *1st Workshop on End-User Software Engineering*, pages 52–56, 2005.

[4] R. Abraham and M. Erwig. AutoTest: A Tool for Automatic Test Case Generation in Spreadsheets. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*, pages 43–50, 2006.

[5] R. Abraham and M. Erwig. Inferring Templates from Spreadsheets. In *28th IEEE Int. Conf. on Software Engineering*, pages 182–191, 2006.

[6] R. Abraham and M. Erwig. Type Inference for Spreadsheets. In *ACM Int. Symp. on Principles and Practice of Declarative Programming*, pages 73–84, 2006.

[7] R. Abraham and M. Erwig. GoalDebug: A Spreadsheet Debugger for End Users. In *29th IEEE Int. Conf. on Software Engineering*, 2007. to appear.

[8] R. Abraham and M. Erwig. UCheck: A Spreadsheet Unit Checker for End Users. *Journal of Visual Languages and Computing*, 18(1):71–95, 2007.

[9] R. Abraham, M. Erwig, S. Kollmansberger, and E. Seifert. Visual Specifications of Correct Spreadsheets. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*, pages 189–196, 2005.

[10] G. Engels and M. Erwig. ClassSheets: Automatic Generation of Spreadsheet Applications from Object-Oriented Specifications. In *20th IEEE/ACM Int. Conf. on Automated Software Engineering*, pages 124–133, 2005.

[11] M. Erwig, R. Abraham, I. Cooperstein, and S. Kollmansberger. Automatic Generation and Maintenance of Correct Spreadsheets. In *27th IEEE Int. Conf. on Software Engineering*, pages 136–145, 2005.

[12] M. Erwig, R. Abraham, S. Kollmansberger, and I. Cooperstein. Gencel — A Program Generator for Correct Spreadsheets. *Journal of Functional Programming*, 16(3):293–325, 2006.

[13] M. Erwig and M. M. Burnett. Adding Apples and Oranges. In *4th Int. Symp. on Practical Aspects of Declarative Languages*, LNCS 2257, pages 173–191, 2002.

[14] M. Fisher, G. Rothermel, D. Brown, M. Cao, C. Cook, and B. Burnett. Integrating Automated Test Generation into the WYSIWYT Spreadsheet Testing Methodology. *ACM Trans. on Software Engineering and Methodology*, 15:150–194, 2006.

[15] G. Rothermel, M. M. Burnett, L. Li, C. DuPuis, and A. Sheretov. A Methodology for Testing Spreadsheets. *ACM Transactions on Software Engineering and Methodology*, pages 110–147, 2001.