How to Communicate Unit Error Messages in Spreadsheets*

Robin Abraham and Martin Erwig
School of Electrical Engineering and Computer Science
Oregon State University
[abraharo|erwig]@cs.orst.edu

Abstract

In previous work we have designed and implemented an automatic reasoning system for spreadsheets, called UCheck, that infers unit information for cells in a spreadsheet. Based on this unit information, UCheck can identify cells in the spreadsheet that contain erroneous formulas. However, the information about an erroneous cell is reported to the user currently in a rather crude way by simply coloring the cell, which does not tell anything about the nature of error and thus offers no help to the user as to how to fix it.

In this paper we describe an extension of UCheck, called UFix, which improves the error messages reported to the spreadsheet user dramatically. The approach essentially consists of three steps: First, we identify different categories of spreadsheet errors from an end-user's perspective. Second, we map units that indicate erroneous formulas to these error categories. Finally, we create customized error messages from the unit information and the identified error category. In many cases, these error messages also provide suggestions on how to fix the reported errors.

Keywords: *Spreadsheet, Program Analysis, Error Messages, End-User Software Engineering*

1 Introduction

Spreadsheet systems like Excel are without doubt the most widely used programming systems. Since spreadsheets are also very likely to contain errors—some studies report that 90% or more of real-world spreadsheets contain errors [6], methods that can improve the level of correctness for spreadsheets can have an enormous positive impact. These methods may aim at detecting errors [2, 3], correcting them [7], or even preventing them [4]. In any case, it is important that these methods can be integrated smoothly into the process of spreadsheet programming, because otherwise they would risk being not accepted by end users.

The *UCheck* system can automatically detect errors in spreadsheet formulas. UCheck works in three phases: First, a spatial analysis determines *header information* for the cells in a spreadsheet [2]. This header information associates labels in the spreadsheet with other cells in the spreadsheet. Based on this header information, a rule system assigns in a second step units to all cells in the spreadsheet [5]. Units can be simple base units that are given by headers, that is, labels of the spreadsheet, but they can also be more complex unit expressions representing *and*, *or*, and *dependent* units. In a final step, UCheck tries to transform the units derived by the rule system into a normal form. This transformation follows a set of equivalences of unit expressions. The error-detection capability of UCheck results from the fact that a cell whose unit expression cannot be simplified to a normal form contains an erroneous formula.

Since UCheck is invoked by simply pressing a button, it requires only minimal effort from the user to obtain a diagnosis about possible errors in a spreadsheet. Moreover, initial tests have indicated that the UCheck system performs accurately in practice [2]. However, a problem of the current UCheck system is that it reports error messages to the user only by coloring cells. Even though primary errors are distinguished from dependent errors[1] by using different colors to focus the user's attention to the important problems in the spreadsheet, no information about the nature of the error is communicated to the user. Thus, even though we can effectively spot erroneous cells, it is not easy for the user to discern what exactly the problem is and how to fix the formulas.

Fortunately, the unit information that has been derived for an erroneous cell contains enough information to reveal more details about the problem of the cell's formula. This fact offers an opportunity to exploit the structure of the derived units to create meaningful error messages that are more helpful to the user than a plain error coloring and may

*This work is supported by the National Science Foundation under the grant ITR-0325273 and by the EUSES Consortium [1].

[1] All dependent errors will disappear from the spreadsheet when all primary errors are corrected.

guide her or him in fixing the spreadsheet.

Our approach to derive error messages consists of the following three steps:

- First, identify error categories and corresponding error messages that are meaningful to the end user. One example is: “The range in the aggregation formula is too small.”
- Second, map the unit structure of the unit indicating the error, which is by definition not in normal form, to one of the error categories. For example, if the unit in a cell has been inferred as the *and* of two non-compatible units and the cell only has a reference to another cell, the system can infer that the error is being caused by the reference.
- Third, compose a concrete error message from the available unit information. For example, if the cause of the unit error has been identified as an incorrect reference, the system can generate an error message to convey this information to the user. Moreover, the system, in many cases, has enough information to generate suggestions that would enable the user to correct the error.

We will describe this approach in more detail in the rest of this paper. In Section 2 we briefly review header and unit inference. A classification of unit errors is presented in Section 3. In Section 4 we show how unit errors are reflected by units that cannot be reduced to normal form and how this information can be used to create error messages. For lack of space, we describe the process of creating error messages by examples. A complete, formal description will be given in a future paper. We present conclusions and plans for future work in Section 5.

2 Identifying Spreadsheet Errors through Unit Inference

In this section, we give a brief and informal overview of header and unit inference. More details can be found in [2, 5].

Consider the spreadsheet shown in Figure 1. We can observe that cell C4 does not just contain a number. In the context of this spreadsheet, it represents the number of oranges harvested in June because of the corresponding row and column headers. The header inference component of the UCheck system determines these relationships between the labels and the cells automatically. The header information is then used to infer the units for the cells. The units of data cells (cells without formulas) are determined directly from the header information. For example, the unit of C4 is inferred as Fruit[Orange]&Month[June]. Here Orange is the column-level header of C4, and Fruit is inferred as the header for Orange. This gives rise to the *dependent* unit Fruit[Orange] for C4. Similarly, the row-level header information leads to the dependent unit Month[June]. Together, these two units are combined to the *and* unit Fruit[Orange]&Month[June].

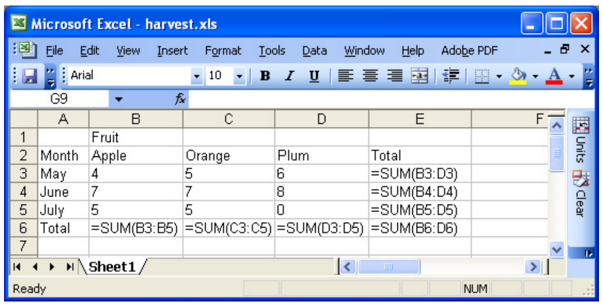
The units of cells with formulas are inferred on the basis of the functions in the formulas. For example, E3 has a formula that sums over the values in cells B3, C3, and D3. Since the units of B3, C3, and D3 are Fruit[Apple]&Month[May], Fruit[Orange]&Month[May], and Fruit[Plum]&Month[May] respectively, the unit of E3 is inferred as

Fruit[Apple]&Month[May]|Fruit[Orange]&Month[May]
|Fruit[Plum]&Month[May]

This unit expression can be simplified by factoring the Month[May] unit to

Fruit[Apple|Orange|Plum]&Month[May],

which then generalizes to Fruit&Month[May].


Microsoft Excel - harvest.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
Arial 10
G9
Sheet1
Ready
NUM
Units
Clear

	A	B	C	D	E	F
1		Fruit				
2	Month	Apple	Orange	Plum	Total	
3	May	4	5	6	=SUM(B3:D3)	
4	June	7	7	8	=SUM(B4:D4)	
5	July	5	5	0	=SUM(B5:D5)	
6	Total	=SUM(B3:B5)	=SUM(C3:C5)	=SUM(D3:D5)	=SUM(B6:D6)	
7						

Figure 1. A unit-correct spreadsheet.

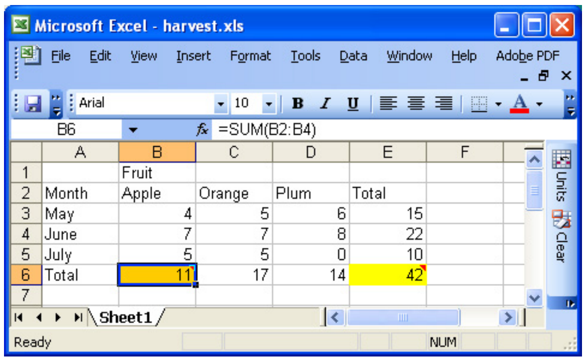
Once the units for all the cells have been inferred, the system checks to see if all the units can be reduced to normal form, as demonstrated for cell E3. In instances in which the units cannot be reduced to normal form, the system reports a unit error.

In the example shown in Figure 2, the range of the formula in cell B6 is offset by one row. In particular, the formula includes a reference to cell B2, which has the header Fruit.

Since cell B2 has the unit Fruit, B3 has the unit Month[May]&Fruit[Apple], and B4 has the unit Month[June]&Fruit[Apple], the unit for cell B6 is inferred as:

Fruit|Month[May]&Fruit[Apple]|Month[June]&Fruit[Apple]

Even though the unit can be partially restructured by factoring Fruit[Apple] into Fruit|Month[May|June]&Fruit[Apple], the unit Month[May|June] cannot be generalized to Month


Microsoft Excel - harvest.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
Arial 10
Units
Clear
Sheet1
Ready
NUM

B6 =SUM(B2:B4)

	A	B	C	D	E	F
1		Fruit				
2	Month	Apple	Orange	Plum	Total	
3	May	4	5	6	15	
4	June	7	7	8	22	
5	July	5	5	0	10	
6	Total	11	17	14	42	
7						

Figure 2. Range-offset error.

because July is missing from the unit. Moreover, the unit cannot be further simplified since Fruit and Fruit[Apple] do not match. Since the unit cannot be transformed into a normal form, a unit error is identified. However, in the UCheck system the user would just see the error as a colored cell. The details about the offset range are not communicated to the user.

3 Categories of End-User Spreadsheet Errors

All the cells in a spreadsheet are assigned units based on their headers. Only cells with formulas can have unit errors. Based on the kinds of formula errors we have in cells, we can classify unit errors as follows. Note that this classification is by no means intended to be general or complete. It has been chosen to help group the different cases of unit errors in order to make it easier to report them to the end user. It is therefore an error classification specifically designed as a frontend for the UCheck system.

1. A **range-too-small error** occurs when the user accidentally excludes one or more cells from a formula.

2. A **range-too-large error** occurs when the formula in a cell refers to the row or column header of that cell.

3. A **range-offset error** occurs when the range in a cell's formula is offset by one or more cells and accidentally includes the row or column header of that cell.

4. An **unexpected-extra-reference error** occurs when the range of the formula in a cell includes cells with incompatible units.

5. A **reference error** occurs when a cell has a reference to another cell with an incompatible unit.

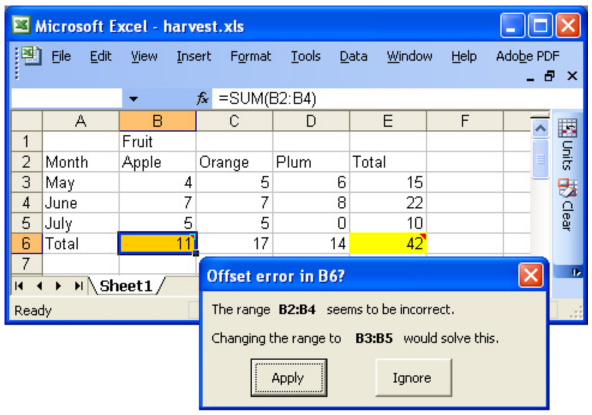
6. An **omission error** occurs when a cell within a formula range is left blank. An omission error is similar to the range-too-small error in that the cell that has a reference to the blank cell still has a valid unit.

In addition to the kind of unit error, we also have to generate information (using terminology the user can understand) about where and how they are manifested in the spreadsheet and how to fix them if possible. In some cases, an error in some cell of the spreadsheet might result in an invalid unit being inferred for some other cell within the spreadsheet. Examples of this scenario are shown in Figures 5 and 7. In such situations, it is important for the system to help the users focus their debugging efforts on the cell that is the cause of the unit error.

4 Creating Error Messages From Unit Structure

In this section we look at cases of units that cannot be reduced to normal form and how they can be mapped to the categories we have described in Section 3. This strategy of mapping non-normal-form units into end-user error messages is currently being implemented as a new front-end to UCheck. It turns out that in many cases the generated error messages can be supplemented by suggestions of how to fix the error. We therefore call this extension of the UCheck system *UFix*.

The example in Figure 3 shows how the error situation shown in Figure 2 will be communicated to the end user in UFix.


Microsoft Excel - harvest.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
Units
Clear
Sheet1
Ready
Apply
Ignore

=SUM(B2:B4)

	A	B	C	D	E	F
1		Fruit				
2	Month	Apple	Orange	Plum	Total	
3	May	4	5	6	15	
4	June	7	7	8	22	
5	July	5	5	0	10	
6	Total	11	17	14	42	
7						

Offset error in B6?

The range **B2:B4** seems to be incorrect.

Changing the range to **B3:B5** would solve this.

Figure 3. Offset error in UFix.

As discussed earlier, the unit for cell B6 is inferred as Fruit|Month[May|June]&Fruit[Apple]. Units of B3 and B4 can be combined to give a valid unit, but the inclusion of B2 in the formula as well as the omission of B5 results in the unit error. The system has already inferred B2 as a header for B3 and B4. This information allows UFix to infer that the reference to B2 is incorrect. Only removing the reference to B2 from the formula would result in a range-too-small error since the formula in B6 would then only have references to B3 and B4. This fact allows the system to identify the error as a likely instance of range-offset error and generate the error message shown in the figure.

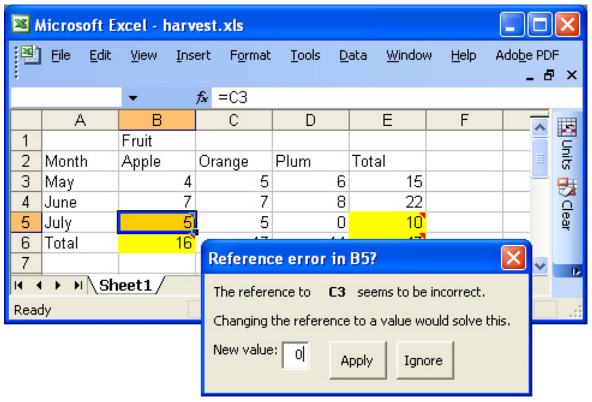
Figure 3 shows the main components of the new error reporting mechanism.

1. The title bar of the error-message window indicates the cell the error has been detected in and the class of the error (this information might prove more useful to users as they become more familiar with the system).
2. The first sentence of the error message explains the problem detected by UCheck/UFix.
3. The second sentence states a proposed solution to the problem.
4. The buttons show the possible user actions. Users can ask the system to make the recommended change by clicking on the "Apply" button. The users can also choose to ignore the generated suggestion by clicking the "Ignore" button.

In the example shown in Figure 4, cell B5 has a reference to cell C3. B5 has the unit Fruit[Apple]&Month[July] because of its position, and the unit Fruit[Orange]&Month[May] because of the reference to C3. The resulting unit is

Fruit[Apple]&Month[July]&Fruit[Orange]&Month[May],

which is not a valid unit since a number cannot represent apples from July *and* oranges from May. The component of the unit that arises from the position of the cell cannot be avoided. Inspection of the erroneous unit indicates that the reference is the part that is causing the error (instance of incorrect reference error).


Microsoft Excel - harvest.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
Sheet1
Ready
Units
Clear

=C3

	A	B	C	D	E	F
1		Fruit				
2	Month	Apple	Orange	Plum	Total	
3	May	4	5	6	15	
4	June	7	7	8	22	
5	July	5	5	0	10	
6	Total	16				
7						

Reference error in B5?

The reference to C3 seems to be incorrect.

Changing the reference to a value would solve this.

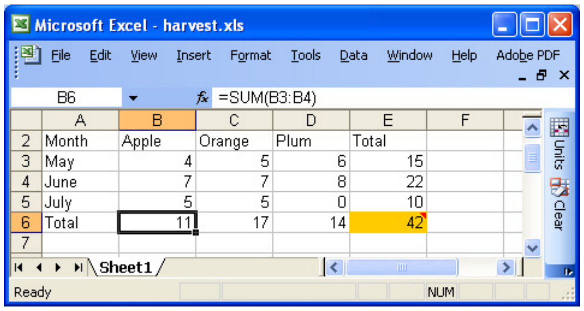
New value: 0

Apply Ignore

Figure 4. Reference error.

It can be formally shown from the unit rule system that any reference from a range that occurs in a SUM formula would lead to a unit error. Therefore, the error message suggests to replace the reference by a value, which can edited within the error window.

In the example shown in Figure 5, the formula in cell B6 references B3 and B4 but not B5. The unit for this formula is inferred as Fruit[Apple]&Month[May|June], which is a valid unit. The units for the cells C6 and D6 are inferred as Fruit[Orange]&Month and Fruit[Plum]&Month, respectively (after generalization). This, in UCheck, results in an error being reported in E6 because its unit cannot be reduced to normal form since the Month[July] component is missing from the unit of B6 (thereby preventing its unit from being generalized to Fruit[Apple]&Month).

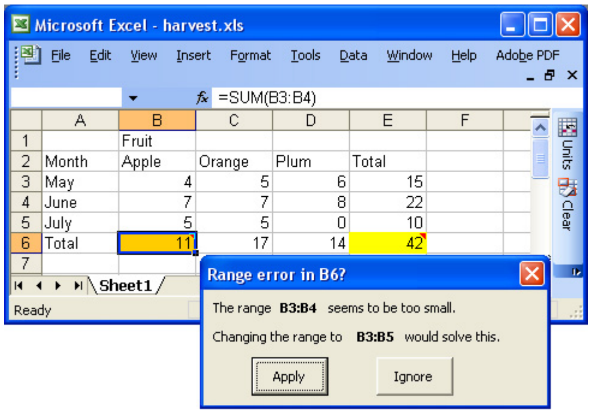

Microsoft Excel - harvest.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
Sheet1
Ready
NUM
Units
Clear

B6 =SUM(B3:B4)

	A	B	C	D	E	F
2	Month	Apple	Orange	Plum	Total	
3	May	4	5	6	15	
4	June	7	7	8	22	
5	July	5	5	0	10	
6	Total	11	17	14	42	
7						

Figure 5. Range-too-small error in UCheck.

UFix inspects the erroneous unit in E6 to identify this case as an instance of range-too-small error and adapts the error message to point to the cell B6 as shown in Figure 6. This example shows that whereas the "coloring of unit error" strategy of UCheck is sometimes wrong about the location of an error, the unit analysis and associated error reporting in UFix gives much more precise information.


Microsoft Excel - harvest.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
Sheet1
Ready
Units
Clear

=SUM(B3:B4)

	A	B	C	D	E	F
1		Fruit				
2	Month	Apple	Orange	Plum	Total	
3	May	4	5	6	15	
4	June	7	7	8	22	
5	July	5	5	0	10	
6	Total	11	17	14	42	
7						

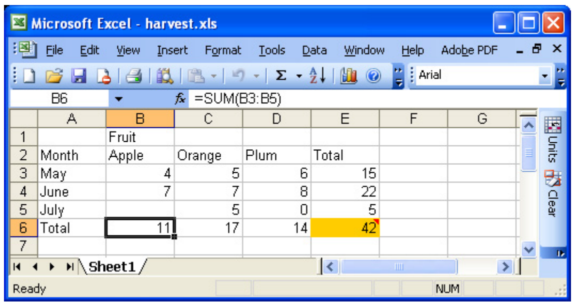
Range error in B6?

The range B3:B4 seems to be too small.

Changing the range to B3:B5 would solve this.

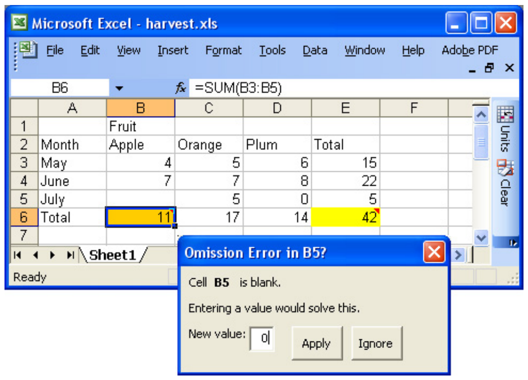
Apply Ignore

Figure 6. Range-too-small error in UFix.


Microsoft Excel - harvest.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
Arial
Units
Clear
Sheet1
Ready
NUM

B6 =SUM(B3:B5)

	A	B	C	D	E	F	G
1		Fruit					
2	Month	Apple	Orange	Plum	Total		
3	May	4	5	6	15		
4	June	7	7	8	22		
5	July		5	0	5		
6	Total	11	17	14	42		
7							


Microsoft Excel - harvest.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
Units
Clear
Sheet1
Ready
Apply
Ignore

B6 =SUM(B3:B5)

	A	B	C	D	E	F
1		Fruit				
2	Month	Apple	Orange	Plum	Total	
3	May	4	5	6	15	
4	June	7	7	8	22	
5	July		5	0	5	
6	Total	11	17	14	42	
7						

Omission Error in B5?

Cell **B5** is blank.

Entering a value would solve this.

New value: 0

Figure 7. Comparison of error messages for omission error in UCheck and UFix.

In our final example shown in Figure 7, B6 has references to cells B3, B4, and B5. Since empty cells are not assigned any units, the reference to B5 does not contribute to the unit of B6. The omission error thus results in an incorrect unit in E6 in the old system along the lines of the range-too-small error discussed above. Even though the erroneous unit in E6 has the same structure in both cases, inspection of the formula in B6 allows UFix to categorize this situation as an instance of omission error and tailor the error message to point the user to the empty cell B5.

In the case of omission error messages (along the lines of the example shown in Figure 7), the message window again allows the user to specify any value for the blank cell. The users can also choose to ignore the generated suggestions by clicking the "Ignore" button.

5 Conclusions and Future Work

We have outlined an approach to interpret structural information about spreadsheets into error messages for end users. Although the error detection is based on a set of formal typing rules and a non-trivial unit structure, users do not have to understand the unit-based reasoning, which happens completely behind the scenes. Moreover, since the unit structure can be mapped to error scenarios that are described in terms of formulas, ranges, etc., user do not even have to understand the notion of units.

The UFix error frontend is currently under development. To facilitate the mapping from units into error messages, we had to refactor the unit-reasoning backend. So far the unit inference was concerned exclusively with identifying correct unit expressions—all other unit expressions were just classified as unit errors.[2] In contrast, we need precise information about the unit structure of erroneous units to enable the identification of error situations and the creating of error messages and change suggestions. We also expect this restructuring to reveal more error situations.

[2]In UCheck we actually distinguish primary sources of errors from secondary ones that depend on primary errors.

Future work will include a formal description of the error-mapping and change-suggestion process. This formalization together with the work on the implementation might reveal more error cases that UFix can successfully identify and report.

We also plan to perform a user study to gather feedback from end users about the usefulness and usability of the system. In the past we have obtained quite useful feedback from teachers who participated in a continuing education event at Oregon State University, which has been organized as part of the efforts of the EUSES consortium [1]. We will continue to use this valuable source of end-user feedback.

References

[1] EUSES: End Users Shaping Effective Software. `http://EUSESconsortium.org`.

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

[3] M. M. Burnett, A. Sheretov, B. Ren, and G. Rothermel. Testing Homogeneous Spreadsheet Grids with the "What You See Is What You Test" Methodology. *IEEE Transactions on Software Engineering*, 29(6):576–594, 2002.

[4] M. Erwig, R. Abraham, I. Cooperstein, and S. Kollmansberger. Automatic Generation and Maintenance of Correct Spreadsheets. *27th IEEE Int. Conf. on Software Engineering*, 2005. To appear.

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

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

[7] J. Ruthruff, E. Creswick, M. M. Burnett, C. Cook, S. Prabhakararao, M. Fisher II, and M. Main. End-User Software Visualizations for Fault Localization. *ACM Symp. on Software Visualization*, pp. 123–132, 2003.