Header and Unit Inference for Spreadsheets Through Spatial Analyses*

Robin Abraham and Martin Erwig
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331, USA
[abraharo|erwig]@cs.orst.edu

Abstract

This paper describes the design and implementation of a unit and header inference system for spreadsheets. The system is based on a formal model of units that we have described in previous work. Since the unit inference depends on information about headers in a spreadsheet, a realistic unit inference system requires a method for automatically determining headers. The present paper describes (1) several spatial-analysis algorithms for header inference, (2) a framework that facilitates the integration of different algorithms, and (3) the implementation of the system.

The combined header and unit inference system is fully integrated into Microsoft Excel and can be used to automatically identify various kinds of errors in spreadsheets. Test results show that the system works accurately and reliably.

1 Introduction

Spreadsheet systems are among the most used software systems. It is estimated that each year tens of millions of professionals and managers create hundreds of millions of spreadsheets [1]. The significance of these numbers becomes clear when put into perspective: The number of end-user programmers, which include spreadsheet users, in the United States alone are expected to reach 55 million by 2005, as compared to only only 2.75 million professional programmers [2].

Numerous studies have shown that existing spreadsheets contain errors at an alarmingly high rate [3, 1, 4, 5]. Some studies even report that 90% or more of real-world spreadsheets contain errors [6]. Thus, due to the widespread use of spreadsheets and their high error rates, there is a huge demand for methods and tools that can improve the reliability of spreadsheets.

Our goal is to enable end users to develop and maintain reliable spreadsheets. To this end we have designed and implemented a *unit reasoning* system that allows end users to identify and correct errors in their spreadsheets. The general idea behind the unit reasoning approach is to exploit information in spreadsheets about labels and headers to check the consistency of cell data and formulas.

In previous work, we have developed a formal reasoning system for detecting unit errors [7]. The unit system uses dependent units, multiple units, and unit generalization to classify the contents of spreadsheets and to check the consistent usage within formulas. Using units, which are based on values in spreadsheets, allows content classification on a more fine-grained level than types do. Moreover, we can communicate with the users in terms of objects contained in the spreadsheet, without having to resort to the abstract concept of types. The advantage of this approach is that it brings the strengths of static type checking to spreadsheets without end users having to pay the cost of learning about type systems.

The unit inference system critically depends on *header information* as input. This information can be provided by the user by means of the techniques discussed in our work on visual customization of inference rules [8]. As an alternative, the related system described in [9] requires the user to completely annotate the value cells with unit information. A principal problem with both of these approaches is that they essentially rely on the user to provide information in addition to the created spreadsheet. However, the user might not be willing to invest the necessary effort to do this [10], especially in the case of larger spreadsheets, or with tight time constraints, or when they extend existing spreadsheets obtained from other users for which the header information might not be obvious. Moreover, the annotation activity might also introduce errors into the spreadsheet.

Therefore, *automatic header inference* seems to be indispensable to make unit inference work in practice. However, the task of automatic header inference is complicated by the fact that spreadsheet systems do not impose any restrictions on the user as far as spatial layout of data is concerned. Moreover, users differ in their preference and style of placing label and header information in spreadsheets.

To solve this problem, we have designed a header-inference framework, which allows us to use a combination of algorithms that infer the header information based on different aspects of the spatial layout of spreadsheets. The framework facilitates the easy extension by new algorithms, and it also allows the adjustment of relative weights given

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

to information obtained by individual algorithms. We have developed and implemented several spatial-analysis algorithms and have assembled them with the help of the framework into an effective header inference system. The system is fully integrated into Excel and will be made available on the Internet shortly. This research is part of the work into end-user programming being performed by the *EUSES* consortium [11].

We present a quick overview of units in Section 2. In Section 3 we demonstrate how units can assist in identifying spreadsheet errors. The header inference framework and the individual spatial-analysis algorithms are described in Section 4. We discuss the system architecture in Section 5. In Section 6 we present some test results for header and unit inference. A discussion of related work follows in Section 7. Future work is outlined in Section 8, and conclusions given in Section 9 complete this paper.

2 Units in Spreadsheets

Units are values in a spreadsheet that describe or label collections of cells, typically (consecutive parts of) rows and columns. Each value in a spreadsheet (except blanks) potentially defines a unit. The unit of any cell is determined by its headers. Intuitively, a header is a label that defines a unit for a group of cells. For example, in Figure 3 Apple is a header for the values in the cells B3, B4, and B5. Units are not just flat entities, but exhibit a certain structure. We essentially have the following kinds of units.

Dependent Units. Since units are values, they can themselves have units; hence, we can get chains of units called *dependent units*. Fruit is a header for Apple and Orange. This hierarchical structure is reflected in our definition of units. In this example, the unit of the cell B3 is not just Apple, but Fruit[Apple]. In general, if a cell *c* has a value *v* as a unit which itself has unit *u*, then *c*'s unit is a *dependent unit* *u*[*v*]. Dependent units are not limited to two levels. For example, if we distinguished red and green apples, a cell containing Green would have unit Fruit[Apple], and a cell whose header is Green would have the dependent unit Fruit[Apple][Green], which is the same as Fruit[Apple[Green]].

And Units. Cells might have more than one unit. For example, the number 11 in cell C3 gives a number of oranges, but at the same time describes a number that is associated with the month May. Cases like this are modeled with *and* units. In our example, C3 has the unit Fruit[Orange]&Month[May].

Or Units. The dual to *and* units are *or* units. *Or* units are inferred for cells that contain operations combining cells of different, but related units. For example, cell D3's formula is B3 + C3. Although the units of B3 and C3 are not identical, they differ only in one part of their *and* unit, Fruit[Apple] and Fruit[Orange]. Moreover, these units differ only in the innermost part of their dependent units. In other words, they share a common prefix that includes the complete path of the dependency graph except the first node. This fact makes the + operation applicable. The unit of D3 is then given as an *or* unit of the units of B3 and C3, that is, Fruit[Apple]&Month[May]|Fruit[Orange]&Month[May].

Not all unit expressions are meaningful. For example, a number cannot represent apples *and* oranges at the same time, although a number can represent apples *or* oranges, that is, fruits. The rules of the unit system define the combination (and simplification) of unit expressions for formulas. Those formulas for which the unit system cannot derive a meaningful unit expression are considered to be incorrect in the sense that they contain a unit error. The following rules define all meaningful unit expressions.

1. Every value that does not have a header is a well-formed unit. For example, in Figure 1, Fruit is a well-formed unit.
2. If a cell has value *v* and header *u*, then it *u*[*v*] is a well-formed unit. For example, in Figure 1, Fruit[Apple] is a well-formed unit.
3. Where there is no common ancestor, it is legal to *and* units. For example, in Figure 1, the unit of B3, Fruit[Apple]&Month[May] is a well-formed unit because Apple and May have no common ancestor.
4. Where there is a common header ancestor, it is legal to *or* units. For example, in Figure 1, Fruit[Apple]|Fruit[Orange], which denotes the same unit as Fruit[Apple|Orange], is well-formed. More precisely, we require that all the values except the most nested ones agree. This is the reason why the unit Fruit[Apple[Green]]|Fruit[Orange] is not well-formed.

A detailed description of units and unit inference can be found in [7].

3 Error Detection with Unit Inference

In the spreadsheet shown in Figure 1, cell B4 contains a reference to cell C3, which has the unit Fruit[Orange]&Month[May]. B4, by virtue of its row and column headers, has the unit Fruit[Apple]&Month[June]. Since B4 contains the reference to cell C3, the system combines both the units and infers the unit for B4 as Fruit[Apple]&Month[June]&Fruit[Orange]&Month[May]. This unit cannot be simplified to a well-formed unit because it violates the third rule for meaningful unit expressions because Apple and Orange have the common ancestor Fruit and so it is illegal to *and* them. Similarly, May and June have the common ancestor Month. Therefore, the system marks cell B4 red to flag a unit error in this cell. Cells B5, D4, and D5 have formulas that have references to cell B4. Since the unit error propagates to these cells, they too are marked red.[1]

This example demonstrates how the unit inference system can detect errors in a spreadsheet that result from wrong

[1] See Section 8 for a a discussion of how to improve the form of visual feedback.

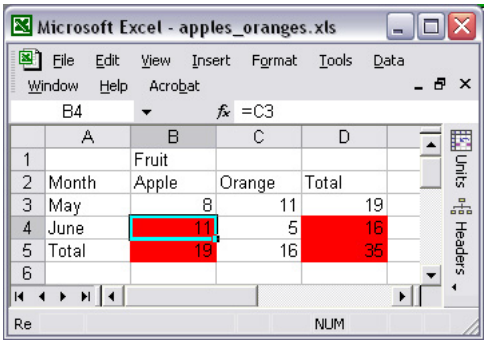

Figure 1. Identified reference errors.

user input, such as overwriting a value by accidentally clicking in a wrong cell.

In the spreadsheet shown in Figure 2, the user has committed an error while entering the formula in B5—instead of finding the sum of cells B3 and B4, the formula tries to compute the sum of cells B2 and B3. Excel ignores this error and returns the result as 8 (the value in cell B3). Furthermore, this error propagates, resulting in an incorrect value in D5 as well since it has a reference to cell B5.

When we run the unit checker on this spreadsheet, the system infers the unit of cell B5 as Fruit|Month[May]&Fruit[Apple] since B2 has the unit Fruit and B3 has the unit Month[May]&Fruit[Apple]. The cell is marked as the site of a unit error since its unit cannot be further simplified to a well-formed unit because Fruit and Month[May]&Fruit[Apple] do not have a common ancestor. So it is illegal to *or* them—violation of rule 4 discussed above. Since the aggregation formula in D5 refers to B5, the unit error propagates and causes D5 to be marked as an error.

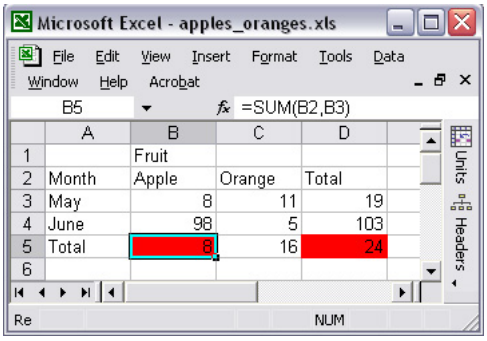

Figure 2. Identified range error.

Again, the unit inference system can identify an error in the spreadsheet, this time a wrong range in a formula.

4 Header Inference

The information about which cells are headers for other cells is crucial for the unit inference. The development of automatic header inference therefore provides the missing link for an automated unit inference system.

The identification of header information is based on the spatial layout of a spreadsheet. Since spreadsheets differ greatly in their layout, it is unlikely that a single algorithm works equally well in all cases. Therefore, we have developed a framework, described in Section 4.1, that allows the integration of different algorithms for spatial analyses. In particular, there are complementary ways of classifying the roles of cells in a spreadsheet. We describe these algorithms briefly in Section 4.2. Based on these cell classifications, the headers are assigned in a multi-level process. These algorithms are explained in Section 4.3. We complete the description of header inference in Section 4.4 with a small case study that demonstrates how the framework helped us to integrate different algorithms.

4.1 Analysis Framework

Header inference is based on the view that a spreadsheet is essentially composed of one or more *tables*. We use the information about the spatial arrangement of cells to classify the cells in a spreadsheet into the following groups.

1. Header: The user uses these to label the data.
2. Footer: These are typically placed at the end of rows or columns and contain some sort of aggregation formula.
3. Core: These are the data cells.
4. Filler: These can be blank cells or cells with some special formatting used to separate tables within the sheet.

We have defined several algorithms that classify spreadsheet cells into the categories mentioned above. Since the algorithms are not equally accurate at identifying the roles of the cells, we assign levels of confidence to the classifications depending on the algorithm used. The confidence levels can range from 1 (minimum confidence) to 10 (maximum confidence). The header inference framework has been designed to allow the easy selection of any combination of algorithms and the weights used by them. Whenever a cell is classified in multiple categories, we sum the confidence levels for each of the categories and pick the classification with the highest sum. This flexibility has allowed us to study the performance and effectiveness of the individual algorithms and in tuning the confidence parameters associated with the algorithms.

4.2 Cell Classification

The following strategies are employed to classify spreadsheet cells.

Fence Identification. A *fence* is a row or a column of cells that form a boundary (upper, lower, left, or right) of

a table. If the fence consists of blank cells, we treat it as a *hard* fence, otherwise, we treat it as a *soft* fence (which is typically the case when the fence consists of repeated headers). Hard fences are classified with a high level of confidence and soft fences are classified with a lower level of confidence.

Content-Based Cell Classification. This algorithm classifies cells as headers, footers, and core simply based on their content. For example, cells with aggregation formulas are classified as footer cells, cells with numerical values are classified as core cells, and cells with string values are classified as header cells. The classification performed by this algorithm is assigned a low level of confidence.

Region-Based Cell Classification. In cases where we have knowledge about the extent of a table (this can be inferred once we have identified fences), we can classify some roles with a higher level of confidence. For example, if the top row or leftmost column of a table is composed of strings, we classify them as headers with a high level of confidence. Similarly, if the last row or rightmost column of a table has aggregation formulas, we classify these as footers.

Footer-to-Core Expansion. In a first step we identify the cells that have aggregation formulas. Such cells are classified as footers with a low level of confidence. We then look at the cells that are referenced by the aggregation formulas (these are the *seed* cells). These are classified as core cells with a high level of confidence. We look at the immediate neighbors of the seed cells. If they are of the same type as the seed cells, they are classified as core cells too. In this way, we use the identified seed cells to grow the core regions. Once we have identified the core and footer cells, we can mark the rest of the cells as header or filler depending on whether or not they have content. This algorithm allows us to identify core cells, headers, and footers.

4.3 Header Assignment

For every core cell, we assign as first-level headers the nearest row (to the left) and column (above) header cells. For example, this would result in cell B4 being assigned Apple and June as headers, see Figure 3.

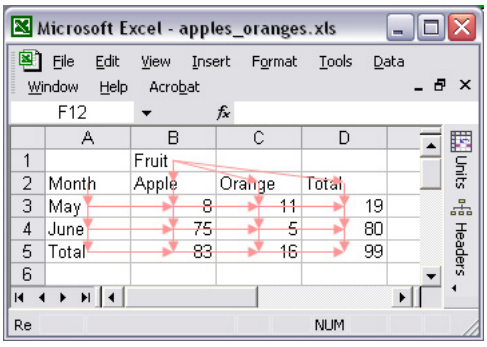

Figure 3. Inferred headers.

Once the first-level headers have been assigned, we can partition the original set of headers into two sets depending on whether or not they have already been assigned to a core cell. Let A be the headers that have been assigned to core cells and U be the set of headers that have not been assigned to any cell yet. Some of the elements of set U might be candidates for higher-level headers whereas others might be just comments.

We impose the following restrictions while inferring higher-level headers.

1. First-level headers cannot act across fences. Higher-level headers can act across fences.
2. An n-level header cannot be assigned as a header for another header at level n.
3. A header at level n can only have one header at level $n+1$. If this rule is violated, the resulting dependent unit would not be well-formed.
4. If two cells are headers for a core cell, they cannot have a common header assigned to them. For example, in Figure 3, the core cell B4 has been assigned Apple (cell B2) and June (cell A4) as headers. If B2 and A4 are assigned a common header, B4 would already have a unit error (violation of Rule 3 in Section 2).

Constraints 3 and 4 essentially limit headers to trees and prevents DAGs.

In addition to the above, we impose the following spatial constraints to exclude user comments from being inferred as headers. Elements of set U that fail the constraints are excluded from the set.

1. We do not assign a higher-level header with only one child since such an assignment would not be of any use from a unit inference point of view.
2. Because of the previous condition, if there are k headers at level n, level $n+1$ can have at most $k/2$ headers. We also require that the headers at level $n+1$ be separated by at least the average distance between the headers at level n. (We will discuss this distance measure in more detail below.)

The headers in set A are either row headers or column headers. Any element $u \in U$ can potentially be a higher-level column header for a subset of column headers $\{a_i | a_i \in A\}$ if the row number of u is less than the row numbers of all a_i's. In other words, a higher-level column header has to be located at the same row level or above the cells it is the header of. Similarly, for higher-level row headers, we require that they are at the same column level or to the left of the cells they are a header of.

In addition to the above conditions, we have a *cost* associated with every assignment of some a_i to u. For column headers, the cost is the column distance between u and a_i and in the case of row headers, the cost is the row distance between u and a_i. Unassigned elements in U receive an

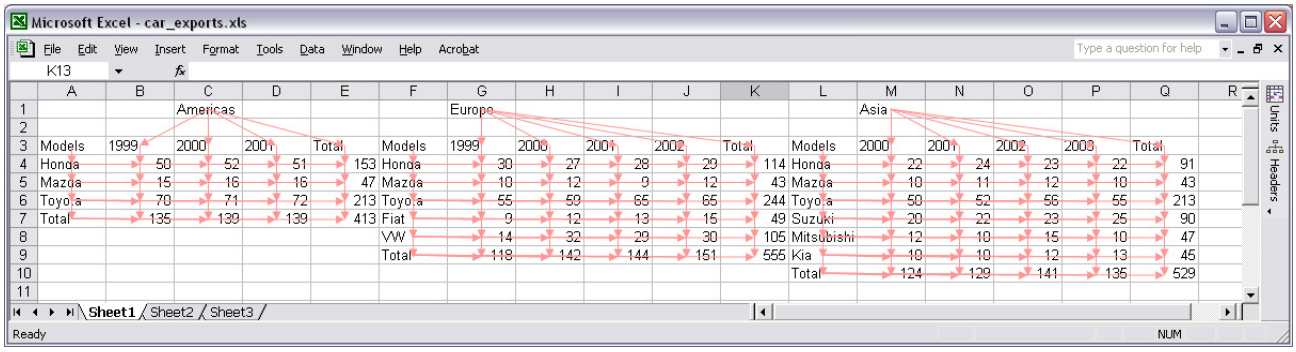

Figure 4. Car exports.

infinite cost to encourage the assignment of all valid higher-level headers. Once the system has generated all the possible combinations, it tries to minimize the overall cost. We first demonstrate how this works using the simple example in Figure 3 and then look at a more complicated case and discuss an extension to our algorithm.

In the example in Figure 3, Apple, Orange, and Total (in D2) are assigned as column headers and May, June, and Total (in A5) are assigned as row headers by the nearest-header algorithm discussed above. This leaves Fruit and Month as the only elements of set U. Fruit cannot be assigned as a row header for cells A3, A4, and A5 because of the spatial constraints discussed above. The cost for assigning Fruit as header for Apple, Orange, and Total is $0+1+2=3$. In contrast, the cost for assigning Month as the column header for Apple, Orange, and Total is 6. Only one of these assignments (either Fruit or Month) can be selected. If the assignment for Month is selected, Fruit will remain unassigned and this would result in an overall infinite cost for the assignment. On the other hand, the cost in assigning Month as row header for May, June, and Total is also 6. This assignment results in an overall cost of 9 since it does not conflict with the assignment of Fruit as column header. Therefore the systems assigns the headers as shown in Figure 3.

We consider a more complicated spreadsheet shown in Figure 4. The spreadsheet contains data for the number of cars exported to North & South America (Americas), Europe and Asia, broken down by makers and years. After the cell classification and the identification of first-level headers, Models is assigned as the second-level row header for the first-level headers under it in columns A, F, and L. To assign the column headers, we exploit the fact that a label can be expected to be in the proximity of (some of) the cells it is describing, because otherwise, it would not serve its purpose. Moreover, people have their own preferences in how they position higher-level headers. For example, in the case of column headers, some people might prefer to position the higher-level header above the first column of subheaders (as we have done in Figure 3 with Fruit), whereas others might prefer to position the higher-level header centered above its subheaders. In the current example, Europe and Asia have been positioned as per the first convention whereas Americas has been more or less centered above its subheaders. The system takes advantage of the spatial information, even when it is not fully consistent, and fences to come up with the correct header inference shown in the figure in the following way.

1. First, we compute the column distance between Americas and Europe. Let this distance be d_1 (4 in this case). Starting at cell B3[2] we traverse distance d_1 to the right and reach cell F3. Based on our discussion above, we consider $d_1 \pm 1$ as a good metric for the level 1 headers that should be made subheaders of Americas. Similarly, we compute the column distance between Europe and Asia, say d_2 (6 in this case). We start at G4 and traverse distance d_2 to the right and reach cell M3. In this case, $d_2 \pm 1$ is a good metric for the level 1 headers that fall under Europe.
2. We have additional clues from the soft fences that are inferred. Since column E is composed of footer cells and column F is composed of header cells, we can infer these as soft fences. Similarly, we can infer soft fences for columns K and L. This information also helps us to correctly assign the higher-level column headers.

The resulting headers are as shown in Figure 4.

4.4 Optimizing Header Inference

While testing the header inference system with the footer-to-core expansion algorithm running at confidence rating 4 and the content-based classification algorithm running at confidence rating 2 we could observe that the system inferred incorrect/insufficient headers for the example spreadsheet shown in Figure 2, as shown in Figure 5.

Because of the incorrect header inference, the system did not detect any unit errors in that case. Since cell B2 participates in the aggregation formula in B5, the footer-to-core

[2] Models in A3 is ignored since it has already been assigned as a second-level header for the other cells in column A. The same is the case with columns F and L.

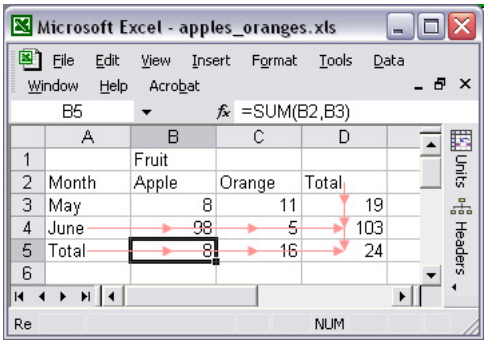

Figure 5. A range error.

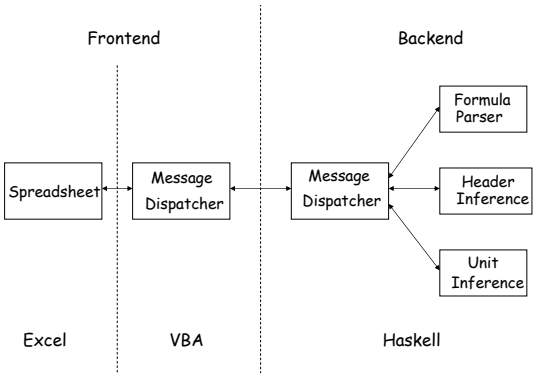

Figure 6. System architecture.

expansion algorithm marks it as a core cell. Furthermore, it checks the neighbors of B2 that have the same type and marks the cells A2, A3, B1, and C2 as core cells (with a confidence level of 4). Since these cells have string values, the content-based classification algorithm marks them as headers with a confidence level of 2. Running only these two algorithms results in the incorrect header inference shown in Figure 5. Increasing the confidence rating of the content-based classification algorithm to a value higher than that of the footer-to-core expansion algorithm would resolve the problem in this particular case, but it would be incorrect in general since the system would then always classify string values as headers.

When the region-based classification algorithm is also enabled, it classifies the first row as a soft fence (one non-blank cell) and the sixth row and column E as hard fences. This results in all the cells in the second row being classified as headers with a confidence rating of 3, the cells in first column being classified as headers with a confidence rating of 5, the cells with formulas in row 5 and column D being classified as footers with a confidence rating of 5. When these classifications are combined with those discussed above, the system comes up with the correct header inference as shown in Figure 3.

5 System Architecture

The header/unit inference engine has been implemented in Haskell. The information from the Excel sheet being manipulated by the end user is captured by a VBA program and sent to the backend server. The VBA system is shipped as an Excel add-in, see Figure 6.

The toolbar has two buttons as shown in the figures. Clicking the "Headers" button displays the header information as inferred by the system. In this view, the system displays arrows directed from the header cells to the target cells as shown in Figure 3. We have enabled this representation for testing purposes so that we can verify the accuracy of the header inference system. In the final version of our system, the user will only see the button for unit checking. When the user clicks the "Units" button, the unit checker is run and the system marks the cells with unit errors with a red background. To communicate from VBA (in Excel) to the server, we use two classes of messages.

1. To send cell data to the server, we use messages of the form `cell row col fml` where `cell` is the keyword recognized by the parser, `row` and `col` have the row and column information respectively, and `fml` is the actual cell content.
2. To send cell formatting information to the server, we use messages of the form `cellF row col fmt` where `cellF` is the keyword that tells the server that the message carries formatting information, `row` and `col` are the row and column information respectively, and `fmt` is the formatting information.

When either the "Units" or the "Headers" button is clicked, the VBA module generates messages for each used cell in the worksheet and sends the messages to the backend engine using a socket connection. The engine parses the messages and builds an internal representation of the Excel worksheet on which it then runs the inference algorithms.

The VBA program receives two classes of messages from the server.

1. *Paint* messages which control display and appearance of the sheet.
2. *Debug* messages which control the display of the information that can be requested while the system is being run in the debug mode. This information helps the developers troubleshoot the system and is not available in the user mode.

6 Evaluation of the System

We have tested our system on two sets of spreadsheets. The first set (set A) consisted of 10 spreadsheet examples from a book by Filby [12] on spreadsheets in science and engineering. The second set (set B) consisted of 18 spreadsheets developed by undergraduate Computer Science students.

Since the output from the header inference algorithms is fed to the unit system, incorrect header inference might result in the system reporting unit errors incorrectly. However, this did not happen with our system. Even in the few cases in which the system came up with slightly incorrect headers, the unit inference did not report any illegal unit errors, because the header inaccuracies occurred for unimportant labels.

Regarding the accuracy of header inference, in set A our system incorrectly reported 4 headers in 1 sheet. In set B, the system reported 3 wrong headers in one sheet and 2 wrong headers in another sheet. As we have mentioned, in no case did an incorrectly inferred header lead to an illegal unit error.

Regarding unit inference, our system detected an omission error in one of the sheets of set A (in the worksheet "P-Cleavage" in workbook "ERTHSCI.XLS"). The same error was detected by the system developed by [9]. This shows that our automatic header inference system works as well as their system, which requires users to manually annotate the cells with unit information. Since this set consists of published spreadsheets, it is not surprising that there are not any more unit errors. In set B, the unit inference system detected errors in 7 sheets. A total of 19 instances of unit errors were detected in set B.

7 Related Work

The pervasiveness of errors in spreadsheets has motivated research into spreadsheet design [13, 14, 15], code inspection [16], quality control [4], testing [17], and consistency checking [7, 8, 18, 9]. Recently, there has also been work on improving the programming capabilities of spreadsheets [19]. This approach follows the guidelines offered by the Cognitive Dimensions of Notations [20] and the Attention Investment model [10].

The use of assertions to identify erroneous formulas is presented in [18]. In this system, the system generates its set of assertions based on the assertions entered by the user. It then warns the user if there is a conflict between the value in the cell and the cell's assertion or when there is a conflict between the system-generated assertion and the user-specified assertion for a cell with a formula. In this context, units can also be considered as a class of system-generated assertions. The main differences between the approaches is that units are automatically inferred and do not constrain the values in the cells.

The system presented in [21] carries out unit checking based on the actual physical or monetary units of the data in the spreadsheet. This approach requires the user to annotate the cells with the unit information, which is then used in the subsequent analysis to flag formulas that violate unit correctness.

The work reported in [9] is most closely related to ours (since it builds on our original work in [7]). Their system supports two kinds of relationships between headers—*is-a* relationship links instances and subcategories whereas *has-a* relationship describes properties of items or sets. Although *has-a* relationships provide more fine-grained information about the headers, they significantly complicate automatic header inference. Accordingly, the approach of [9] requires the user to manually annotate spreadsheets with header information, which is a big drawback. The described unit inference rules are also different from ours, and in trying to be more flexible, the system fails to detect some errors. For example, our system requires all subunits of a unit to be present in an *or* unit expression to be able to generalize. This constraint prevents, for example, the comparison of a number representing apples or oranges with one that represents oranges or bananas. However, the rules described in [9] do not use this constraint and allow the generalization of either *or* unit to fruits and thus also allow the illegal comparison.

8 Future Work

First, we can add more analysis algorithms to further improve the accuracy of cell classification. In this context, all kinds of formatting information provides a valuable source of input since it is often used to emphasize semantically relevant spatial arrangements. The existing framework allows the easy extension by algorithms that exploit formatting information.

In the current implementation, all cells with unit errors are marked with a red background as shown in Figure 1. This can be further refined when we consider the fact that a cell can be assigned an invalid unit in two ways.

1. Local unit error: The unit of the cell under consideration is itself incorrect.
2. Propagation unit error: The cell contains a unit error because it references a cell containing a unit error.

The system can provide the end user with more feedback for fault localization if the cells with local unit errors are marked with a dark red background and the cells with propagation unit errors are marked with a lighter shade of red as shown in Figure 7.

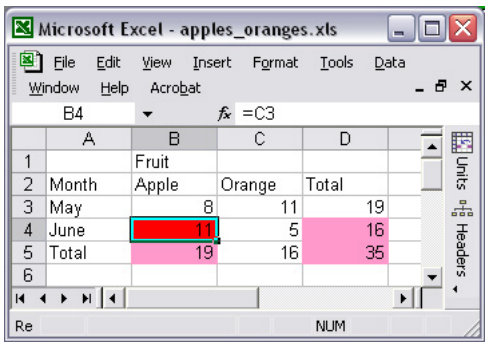

Figure 7. Fault localization feedback.

This would make it easier for to isolate the cause of the unit error. We are also exploring better ways to communicate with the end user based on the SER framework [22].

In the current version of the system users need to click the "Units" button to run the unit inference system. The system can be easily extended (using cell and worksheet-level events available in Excel VBA) to detect changes to the spreadsheet and report those changes to the inference engine. This would enable the system to provide the user with immediate feedback. We are also looking at ways to support incremental header/unit inference. This requires a dependency analysis, but could speed up the response time, which is currently quite fast (subsecond response times for sheets with up to 300 cells). However, giving the user immediate feedback after every action might be too intrusive. We are currently exploring ways by which we can prevent the system from being "too eager" and giving feedback only when the user has completed all the actions involved in a "transaction".

9 Conclusions

We have designed and implemented a system that automatically infers header information in spreadsheets, performs a unit analysis, and informs the user when unit errors are detected. A very important feature of the system is that it does not require the user to provide any extra information, and it runs on any spreadsheet. We have tested our system on spreadsheets collected from two sources of very different expertise level and found that it is working accurately: It has successfully detected the unit errors that are present in the sheets and did not reports any false unit errors.

References

[1] R. R. Panko. Spreadsheet Errors: What We Know. What We Think We Can Do. *Proceedings of the Spreadsheet Risk Symposium, European Spreadsheet Risks Interest Group (EuSpRIG)*, 2000.

[2] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, K. C. Bradford, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece, editors. *Software Cost Estimation with COCOMO II*. Prentice-Hall International, Upper Saddle River, NJ, 2000.

[3] T. Teo and M. Tan. Quantitative and Qualitative Errors in Spreadsheet Development. In *30th Hawaii Int. Conf. on System Sciences*, pages 25–38, 1997.

[4] K. Rajalingham, D. Chadwick, B. Knight, and D. Edwards. Quality Control in Spreadsheets: A Software Engineering-Based Approach to Spreadsheet Development. *33rd Hawaii International Conference on System Sciences*, pages 1–9, 2000.

[5] M. Tukiainen. Uncovering Effects of Programming Paradigms: Errors in Two Spreadsheet Systems. *12th Workshop of the Psychology of Programming Interest Group (PPIG)*, pages 247–266, 2000.

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

[7] 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.

[8] M. M. Burnett and M. Erwig. Visually Customizing Inference Rules About Apples and Oranges. In *2nd IEEE Int. Symp. on Human-Centric Computing Languages and Environments*, pages 140–148, 2002.

[9] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi. A Type System for Statically Detecting Spreadsheet Errors. In *8th IEEE Int. Conf. on Automated Software Engineering*, pages 174–183, 2003.

[10] A. Blackwell. First Steps in Programming: A Rationale for Attention Investment Models. *IEEE Symp. on Human-Centric Computing Languages and Environments*, pages 2–10, 2002.

[11] M. M. Burnett, M. Erwig, M. Niess, and G. Rothermel. EUSES: End Users Shaping Effective Software. `http://eecs.oregonstate.edu/EUSES/`.

[12] G. Filby. *Spreadsheets in Science and Engineering*. Springer, 1995.

[13] T. Isakowitz, S. Schocken, and H. C. Lucas, Jr. Toward a Logical/Physical Theory of Spreadsheet Modelling. *ACM Transactions on Information Systems*, 13(1):1–37, 1995.

[14] A. G. Yoder and D. L. Cohn. Real Spreadsheets for Real Programmers. *Int. Conference on Computer Languages*, pages 20–30, 1994.

[15] B. Ronen, M. A. Palley, and H. C. Lucas, Jr. Spreadsheet Analysis and Design. *Communications of the ACM*, 32(1):84–93, 1989.

[16] R. R. Panko. Applying Code Inspection to Spreadsheet Testing. *Journal of Management Information Systems*, 16(2):159–176, 1999.

[17] 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.

[18] M. M. Burnett, C. Cook, J. Summet, G. Rothermel, and C. Wallace. End-user Software Engineering with Assertions. *25th Int. Conference on Software Engineering*, 2003.

[19] S. L. Peyton Jones, A. Blackwell, and M. M. Burnett. A User-Centered Approach to Functions in Excel. In *ACM Int. Conf. on Functional Programming*, pages 165–176, 2003.

[20] A. F. Blackwell and T. R. G. Green. Notational Systems - The Cognitive Dimensions of Notations Framework. *HCI Models, Theories, and Frameworks: Toward and Interdisciplinary Science*, pages 103–133, 2003.

[21] T. Antoniu, P. A. Steckler, S. Krishnamurthi, Neuwirth, and M. Felleisen. Validating the Unit Correctness of Spreadsheet Programs. In *Int. Conf. on Software Engineering*, 2004.

[22] S. Prabhakarao, C. Cook, J. Ruthruff, E. Creswick, M. Main, M. Durham, and M. Burnett. Strategies and Behaviors of End-User Programmers with Interactive Fault Localization. *IEEE Symp. on Human-Centric Computing Languages and Environments*, pages 203–210, 2003.