UCheck: A Spreadsheet Type Checker for End Users[*]

Robin Abraham and Martin Erwig

School of EECS
Oregon State University
[abraharo, erwig]@eecs.oregonstate.edu

June 12, 2006

Abstract

Spreadsheets are widely used, and studies have shown that most end-user spreadsheets contain non-trivial errors. Most of the currently available tools that try to mitigate this problem require varying levels of user intervention. This paper presents a system, called UCheck, that detects errors in spreadsheets automatically. UCheck carries out automatic header and unit inference, and reports unit errors to the users. UCheck is based on two static analyses phases that infer header and unit information for all cells in a spreadsheet.

We have tested UCheck on a wide variety of spreadsheets and found that it works accurately and reliably. The system was also used in a continuing education course for high school teachers, conducted through Oregon State University, aimed at making the participants aware of the need for quality control in the creation of spreadsheets.

Keywords: Spreadsheet, Unit, Type, Automatic error detection, Debugging, End-user software engineering

1 Introduction

Studies have shown that each year, tens of millions of professionals and managers create hundreds of millions of spreadsheets [26], thereby making spreadsheets the most widely used programming environment [34]. End-user programmers do not have a sound background in software design, development, or maintenance. As a result, existing spreadsheets contain errors at an alarmingly high rate [37, 26, 29, 38]. Some studies even report that 90% or more of real-world spreadsheets contain errors [30]. In many instances, spreadsheet errors have resulted in huge financial losses to the organizations concerned [11, 19, 20, 21, 24, 36, 35]. Many of these "horror stories" have been documented at [16].

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 called UCheck 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.

UCheck basically consists of two components.

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

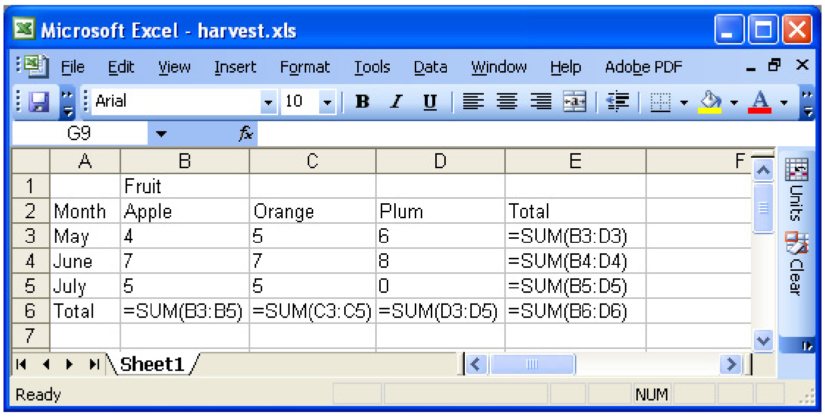
1. A formal reasoning system for detecting unit errors that is based on header information for a spreadsheet [14].

2. A header inference system that automatically determines header information for spreadsheets [1].

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.

Consider, for example, the spreadsheet 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 column and row headers. These kinds of relationships between labels and cells are determined automatically by the system.

Another example is the cell E4. Its unit is inferred to be June and Fruit. This inference is made from the fact that the formula contains a SUM function and aggregates over Apple, Orange, and Plum, which are all identified as being labeled Fruit. Now if the range in the SUM formula was A4:D4 instead, the formula would be aggregating over Apple, Orange, Plum, and Month (the unit of cell A4 is inferred as Month). Since these units do not have a common higher-level unit, this operation would result in a unit error.


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

	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: Inferring units from headers.

As shown by these two examples, 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 [10]. As an alternative, the related system described in [4] 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 [6], 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 to information obtained by individual algorithms. We have developed and implemented several spatial-analysis algorithms and have integrated them with the help of the framework into UCheck. For example, for the spreadsheet in Figure 1, Orange will be inferred as a header for cells C3, C4, C5, and C6 because of its position at the top of the column (C) of numbers. Fruit in B2 on the other hand will be inferred as a header for Apple, Orange, and Plum since it is located above them in the spreadsheet.

The relationships of the different components of UCheck are summarized in Figure 2.

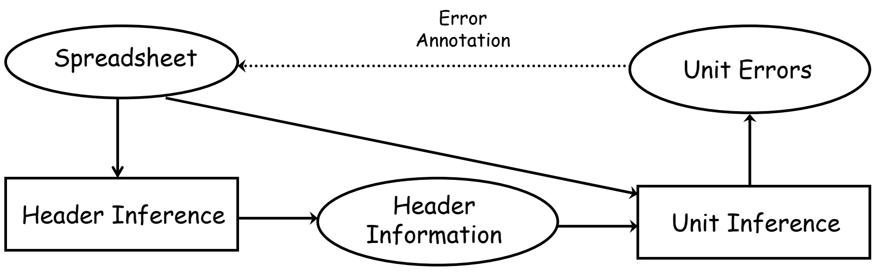


Figure 2: Overview of unit checking.

This research is part of the work into end-user programming being performed by the EUSES consortium [15]. We have already mentioned the impact of spreadsheet errors has been extensively documented. We present a few examples where the use of UCheck could have helped detect the error and prevent losses to the organizations concerned.

1. The illegal interpretation of a date as a numeric value caused an operating fund of the Colorado Student Loan Program to be understated by \$36,131 [39].

2. A reference error caused a hospital's records to overstate its Medicaid/Medicare crossover log by \$38,240 [40].

3. Copy-paste operations introduced errors into the budget proposal submitted by the Lincoln County Communications Agency (LinCom) [19].

4. At Solution Matrix, incorrect parenthesis in a spreadsheet resulted in the projected profits being computed as \$200 million when they actually were only \$25 million [24].

In the next section, we introduce the concept of units in spreadsheets. Section 3 presents the steps involved in automatic header inference. In Section 4, we describe how units can be used to detect errors within spreadsheets. We briefly describe the system architecture of UCheck in Section 5. Steps taken to evaluate our approach are described in Section 6. Related work is presented in Section 7, followed by future work in Section 8. We finish the paper with conclusions given in Section 9.

2 Unit Inference in Spreadsheets

The unit inference algorithm takes the content of the spreadsheet being checked and its header information as input and assigns units for all the cells. The system then tries to simplify the unit information to a normal form that indicates well-formed units. Those cells that have units that cannot be reduced to well-formed units are then reported as sites of unit error. Figure 3 summarizes the structure of the unit inference process. It refines the "Unit Inference" box from Figure 2.

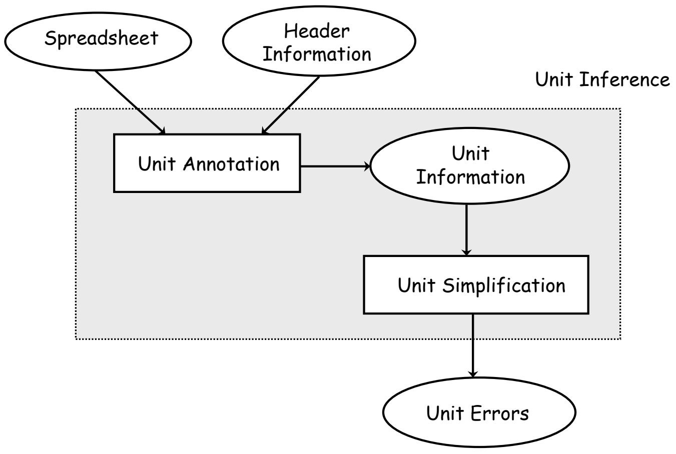


Figure 3: Detecting unit errors.

We present the concept of units informally in the next section and follow it up with a formal discussion of the unit system in Section 2.2. We demonstrate the application of the unit rules using an example in Section 2.3.

2.1 Unit Expressions

We have seen some examples of units in Section 1. A closer look reveals that 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 headers; hence, we can get chains of headers resulting in *dependent units*. Fruit is a header for Apple, Orange, and Plum. 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 header u, then c's unit is a *dependent unit* $u[v]$.

And Units. Cells might have more than one unit. For example, the number 7 in cell C4 gives a number of oranges, but at the same time describes a number that is associated with the month June. Cases like this are modeled with *and* units. In our example, C4 has the unit Fruit[Orange]&Month[June].

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 E3's formula is SUM(B3:D3), which is equivalent to B3 + C3 + D3. Although the units of B3, C3, and D3 are not identical, they differ only in one part of their *and* unit, Fruit[Apple], Fruit[Orange], and Fruit[Plum]. 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 E3 is then given as an *or* unit of the units of B3, C3, and D3, that is, Fruit[Apple]&Month[May]|Fruit[Orange]&Month[May]|Fruit[Plum]&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 *or* plums, 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. In contrast, Fruit[Apple]&Fruit[Orange] is not well-formed.

4. Where there is a common header ancestor, it is legal to *or* units. For example, in Figure 1, Fruit[Apple]|Fruit[Orange]|Fruit[Plum] , which denotes the same unit as Fruit[Apple|Orange|Plum], 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.

2.2 A Formal Unit System

In this section we briefly describe the formal background of the unit system. The described unit system is based on the formal model first introduced in [14]. The most important change is a more sophisticated treatment of generalization using a new concept of "proxy" headers.

We define a simple model of spreadsheets in Figure 4. A spreadsheet (s) is a list of tuples of cell addresses (a) and expressions (e). Expressions can evaluate to values of different types. Two special cases are the error value (ϵ) and the blank value ($\sqcup$) which have been introduced since spreadsheets often contain blanks and errors. Expressions can also be references (a) or operations (ω). The operations are indexed by the number of required arguments. Hence, ω^0 ranges over all constants that are different from $\sqcup$ and ϵ. We use v as a synonym for ω^0 whereas $v^{\sqcup}$ ranges over all constants and $\sqcup$, and v^{ϵ} ranges over all constants, including $\sqcup$ and ϵ. The cell addresses in a spreadsheet are distinct. Thus we can regard a spreadsheet as a mapping from names to expressions. We use $dom(s)$ and $rng(s)$ to denote a spreadsheet's domain and range, respectively.

$$\begin{array}{lcll}
a & & & \text{Names} \\
e & ::= & \sqcup \mid \epsilon \mid \omega^n(e_1,\ldots,e_n) \mid a & \text{Expressions } (n \geq 0) \\
s & ::= & (a_1,e_1)\,;\,\ldots\,;\,(a_m,e_m) & \text{Spreadsheets } (m \geq 1) \\
 & & i \neq j \Rightarrow a_i \neq a_j &
\end{array}$$

Figure 4: Abstract syntax of spreadsheets.

The formal rules for unit inference are shown in Figure 5. We need to consider three kinds of judgments: First, we have judgments of the form $(a,e) :: u$ that associate units to cells and that can exploit header information. Second, for the unit inference for operations we also need judgments $e : u$ that give units for expressions regardless of their position (context). Third, for expressing the units of a spreadsheet we need a further kind of judgment $(a_1, e_1) ; \ldots ; (a_m, e_m) ::: (a_1, u_1) ; \ldots ; (a_m, u_m)$, which expresses the fact that the unit information for a spreadsheet is given by a mapping of addresses to units. Note that the spreadsheet (s) and the header definition (h) are considered global constants in the rules. The rules EQ$_{:}$ and EQ$_{::}$ are based on the equations shown in Figure 8, which define equality of units. Moreover, the premises $\vdash u$ ensure that only well-formed units, as defined in Figure 7, will be derived. For example, VAL$_{::}$ expresses that all cells that do not have a header have the unit $\mathbf{1}$. In the

$$\text{VAL}_{:} \quad \frac{}{v^{\sqcup} : \mathbf{1}} \qquad \text{REF}_{:} \quad \frac{(a, s(a)) :: u \qquad \vdash u}{a : u} \qquad \text{EQ}_{:} \quad \frac{e : u \qquad u =_u u'}{e : u'}$$

$$\text{APP}_{:} \quad \frac{e_i : u_i \qquad \vdash u_i}{\omega^n(e_1, \ldots, e_n) : \mu_\omega(\mathbf{1}, u_1, \ldots, u_n)}$$

$$\text{VAL}_{::} \quad \frac{a \notin dom(h)}{(a, v^{\sqcup}) :: \mathbf{1}} \qquad \text{REF}_{::} \quad \frac{(a, s(a)) :: u \quad (a', s(a')) :: u' \quad \vdash u \quad \vdash u'}{(a, a') :: u \& u'}$$

$$\text{APP}_{::} \quad \frac{e_i : u_i \qquad \vdash u_i \qquad (a, \sqcup) :: u \qquad \vdash u}{(a, \omega^n(e_1, \ldots, e_n)) :: \mu_\omega(u, u_1, \ldots, u_n)} \qquad \text{EQ}_{::} \quad \frac{(a, e) :: u \quad u =_u u'}{(a, e) :: u'}$$

$$\text{DEP}_{::} \quad \frac{h(a) = \{a_1, \ldots, a_k\} \qquad s(a_i) = v_i^{\sqcup} \qquad (a_i, v_i^{\sqcup}) :: u_i \qquad \vdash u_i}{(a, v^{\sqcup}) :: u_1[v_1^{\sqcup}] \& \ldots \& u_k[v_k^{\sqcup}]}$$

$$\text{SHEET}_{:::} \quad \frac{i \in J_m \qquad (a_i, e_i) :: u_i \qquad \vdash u_i \qquad j \in \bar{J}_m \qquad u_j = \epsilon}{(a_1, e_1) ; \ldots ; (a_m, e_m) ::: (a_1, u_1) ; \ldots ; (a_m, u_m)}$$

$$\text{PROXY} \quad \frac{s(b) = \omega^n(e_1, \ldots, e_n) \qquad h(b) = \{a\} \qquad (a, s(a)) :: u}{\wp(a, u)}$$

Figure 5: Unit inference rules.

spreadsheet shown in Figure 1, cells Fruit and Month have unit $\mathbf{1}$. According to DEP$_{::}$, if cell a has a header cell b that contains a value v and has a well-formed unit u, then a's unit is $u[v]$. More generally, if a cell has multiple headers with values v_i and units u_i, then its unit is given by the *and* unit of all the $u_i[v_i]$. An example is the cell B3 containing 4 whose unit is Fruit[Apple]&Month[May]. REF$_{::}$ is applicable if cell a's formula is a reference to cell b. In this case, b's unit, say u_b is propagated to a. If a has itself a header, say u_a, then u_a must conform with u_b, which is achieved by defining a's unit to be $u_a \& u_b$. APP$_{::}$ is applicable in cases cells have formulas with operators. Each operator (μ_ω) has its own definition of how units of its parameters combine. The definitions of unit combinations are shown in Figure 6 for sum (μ_+), count (μ_{count}), product (μ_*), and division ($\mu_/$). (For the sake of convenience, we abbreviate $u_1, \ldots, u_n$ by $\bar{u}$ in Figure 6.) For example, the rule for μ_+ and μ_{count} state that those operations result in the units of the operands being combined together by &. PROXY is applicable to cells that have aggregation formulas. The $\wp(a, u)$ judgment expresses that the cell with address a is a proxy header for u. For example, in the spreadsheet shown in Figure 1, cell E3 has the header

Total (from cell E2) which is just a proxy for Fruit. This inference can be carried out since the aggregation formula in E3 references B3, C3, and D3, thereby including all the headers (Apple, Orange, and Plum) Fruit is the parent of.

$$
\begin{aligned}
\mu_{+}(u,\bar{u}) &= (u_1|\ldots|u_n)\&u \\
\mu_{\mathsf{count}}(u,\bar{u}) &= (u_1|\ldots|u_n)\&u \\
\mu_{*}(u,\bar{u}) &= \downarrow(\bar{u})\&u \\
\mu_{/}(u,\bar{u}) &= u_1\&u \\
\ldots
\end{aligned}
\qquad
\downarrow(\bar{u}) = \begin{cases} u_i & \text{if } u_i \neq \mathbf{1} \wedge \forall j \neq i : u_j = \mathbf{1} \\ \mathbf{1} & \text{otherwise} \end{cases}
$$

Figure 6: Unit transformations.

Well-formed units are defined by the judgments in Figure 7. As in the case for the rules for unit inference, we assume that the spreadsheet (s) and the header information (h) are globally available. The shown rules formalize the conditions described earlier in Section 2.1.

$$
\text{ONE}_{\vdash}\ \frac{}{\vdash \mathbf{1}}
\qquad\qquad
\text{VAL}_{\vdash}\ \frac{a \notin \mathit{dom}(h) \qquad s(a) = v}{\vdash \mathbf{1}[v]}
$$

$$
\text{DEP}_{\vdash}\ \frac{a' \in h(a) \qquad s(a) = v \qquad \vdash u \qquad \vdash u[s(a')]}{\vdash u[s(a')[v]]}
$$

$$
\text{AND}_{\vdash}\ \frac{\vdash \mathbf{1}[u_1[\ldots u_{n-1}[u_n]\ldots]] \qquad \vdash \mathbf{1}[u'_1[\ldots u'_{m-1}[u'_m]\ldots]] \qquad u_i \neq_u u'_i}{\vdash u_1[\ldots u_{n-1}[u_n]\ldots]\&u'_1[\ldots u'_{m-1}[u'_m]\ldots]}
$$

$$
\text{OR}_{\vdash}\ \frac{\vdash \mathbf{1}[u_1[\ldots u_n[v_1]\ldots]] \ \ldots\ \vdash \mathbf{1}[u_1[\ldots u_n[v_k]\ldots]] \quad n > 1 \quad k > 1 \quad v_i \text{ distinct}}{\vdash \mathbf{1}[u_1[\ldots u_n[v_1|\ldots|v_k]\ldots]]}
$$

Figure 7: Well-formed units.

Application of operations can result in arbitrarily complex unit expressions that do not always meet the conditions for well-formed units. Figure 8 shows the properties of well-formed units that allows us to simplify complex unit expressions. Whenever simplification to a well-formed unit is possible, it can be concluded that the operation is applied in a "unit-correct" way. Otherwise, a unit error is reported. Note that & distributes over |, but not vice versa, and that although **1** is the identity unit for &, it leads to a non-valid unit when combined with |. The condition (*) for generalization is that the *or* unit expression consists exactly of all non-proxy units $u_1, \ldots, u_n$ that have u as a header. Moreover, this condition must hold for all definitions (copies) of u (that is, for all cells containing u):

$$
\forall a \in s^{-1}(u) : h^{-1}(\{a\}) = \{a_1, \ldots, a_k, a_{k+1}, \ldots, a_{k+m}\} \ \wedge\ s(a_i) = u_i \ (\text{for } 1 \leq i \leq k) \wedge
$$
$$
\wp(a_i, u) \ (\text{for } k+1 \leq i \leq k+m)
$$

For example, cell E3 in Figure 1 has the unit Month[May]&Fruit[Apple|Orange|Plum]. This unit can be generalized to Month[May]&Fruit since the *or* unit expression has all units Apple, Orange, and Plum, that have Fruit as their header.

2.3 An Example for Unit Checking

We demonstrate the unit inference rules and the unit simplifications with the example spreadsheet from Figure 1 which we call *Harvest*. Since B1 has no header definition, Fruit has unit **1**

$$
\begin{array}{rll}
u_1\&u_2 =_u u_2\&u_1 & & \text{(commutativity)}\\
u_1|u_2 =_u u_2|u_1 & & \\
(u_1\&u_2)\&u_3 =_u u_1\&(u_2\&u_3) & & \text{(associativity)}\\
(u_1|u_2)|u_3 =_u u_1|(u_2|u_3) & & \\
u\&(u_1|u_2) =_u (u\&u_1)|(u\&u_2) & & \text{(distributivity)}\\
u\&u =_u u & & \text{(idempotency)}\\
u|u =_u u & & \\
\mathbf{1}\&u =_u u & & \text{(unit)}\\
u[u_1]|u[u_2] =_u u[u_1|u_2] & & \text{(factorization)}\\
u[u_1|\ldots|u_k] =_u u & \Leftarrow (*) & \text{(generalization)}\\
(u_1[u_2])[u_3] =_u u_1[u_2[u_3]] & & \text{(linearization)}
\end{array}
$$

Figure 8: Unit equality.

by rule VAL$_{::}$, that is (B1, Fruit) :: **1**. Similarly, the unit for Month in cell A2 is inferred as **1**. By rule DEP$_{::}$ we know that Apple has the unit **1**[Fruit], or (B2, Apple) :: **1**[Fruit], because

- the header of B2 is B1,
- the value of the cell B1 is Fruit (that is, *Harvest*(B1) = Fruit), and
- cell Fruit has unit **1**.

Since all units except **1** are of the form **1**[...] we omit the leading **1** for brevity in the following. Hence we also write (B2, Apple) :: Fruit. Similarly, we can reason that (A3, May) :: Month. Finally, using the last two results and the fact that the header of B3 is defined to be the two cells A3 and B2 we can conclude by rule DEP$_{::}$ that (B3, 4) :: Month[May]&Fruit[Apple].

Next we infer the unit for the cell E3. First, we have to infer the units (C3, 5) :: Month[May]&Fruit[Orange] and (D3, 6) :: Month[May]&Fruit[Plum], which can be obtained in the same way as the unit for the cell B3. To infer the unit of E3 we apply rule APP$_{::}$. We already have the units for the three arguments. We can apply μ_+ and obtain as a result an *or* unit for E3, namely:

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

We can now perform rule simplification several times, first exploiting distributivity, which yields the unit:

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

Then we apply factorization, which yields the unit:

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

The aggregation formula in E3 has references to B3, C3, and D3, which have Apple, Orange, and Plum as their headers respectively. Total in E2 has Fruit as its unit. We can apply the PROXY rule to conclude that Total is simply a proxy header for Fruit and can be ignored in the condition for generalization. Therefore using generalization, we obtain the following unit for E3:

Month[May]&Fruit

Now consider a cell with an expression +(B3, C4). The unit for this cell would be inferred as Month[May]&Fruit[Apple]|Month[June]&Fruit[Orange]. Since we get two different months and two different fruits, the application of the distributivity rule is not possible, which prevents the unit expression from being reduced to a well-formed unit. Since we cannot infer a unit in normal form, the system reports a unit error for that cell.

3 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 steps involved in carrying out automatic header inference are shown in Figure 9, which refines the "Header Inference" box from Figure 2.

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 3.1, 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 3.2. Based on these cell classifications, the headers are assigned in a multi-level process. These algorithms are explained in Section 3.3. We complete the description of header inference in Section 3.4 with a small case study that demonstrates how the framework helped us to integrate different algorithms.

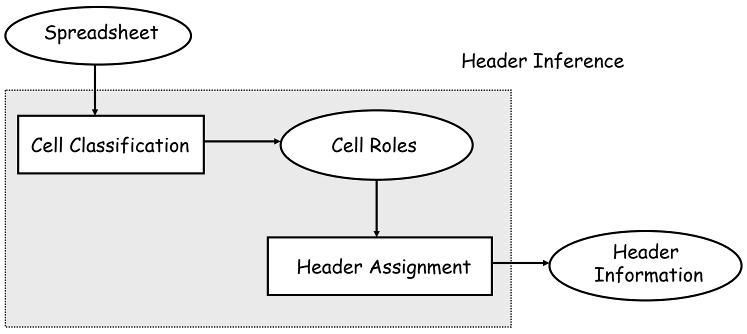


Figure 9: Inferring header information.

3.1 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.

3.2 Cell Classification

In the absence of errors in the spreadsheet, a preliminary classification of the cells based on their roles is conceivable. We can assign cell roles by analyzing the formulas as follows.

- Cells that have formulas and are not referred to by other cells can be classified as footers.
- Cells that do not have formulas and are used strictly as inputs to other cells can be classified as core cells.
- Cells that neither have formulas nor are used as inputs to formulas in other cells can be considered headers.
- Cells that have formulas that reference core cells or other formula cells can be classified as footers as well.

This rather simplistic classification of the spreadsheet cells fails when we have errors in the spreadsheet formulas. That is why we have adopted a heuristics-based approach. 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, whereas 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. Whenever we have knowledge about the extent of a table, which is the case, for example, after 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.

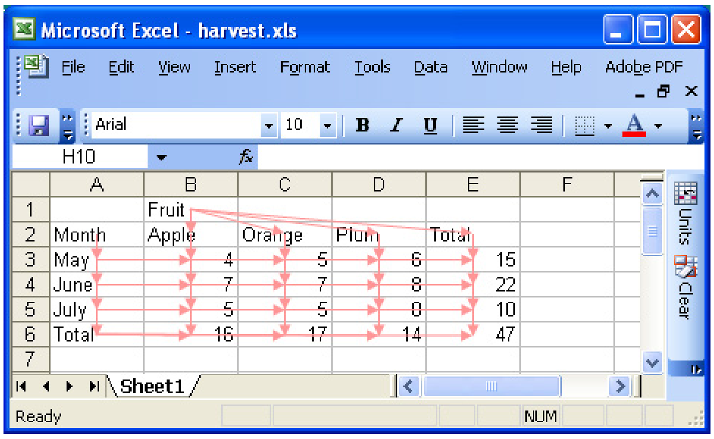
Footer-to-Core Expansion. In a first step we identify the cells that have aggregation formulas. Such cells are classified as footers with a low level of confidence. We then look at the cells that are referenced by the aggregation formulas and classify them as core cells with a high level of confidence. These cells are called *seed cells.* After that 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.

3.3 Header Assignment

After the classification has been carried out, we can assign headers to the core and footer cells. Header assignment is carried out in two steps.

1. As the first step, we assign the *first-level* headers for the core cells.
2. We then assign *higher-level* headers which are basically headers of the first-level headers.

For every core cell, we assign as first-level headers the nearest row (to the left) and column (above) header cells. For example, this strategy results in cell B4 being assigned Apple and June as headers, see Figure 10. The other first-level headers will be Orange, Plum, Total (in cell E2), May, July, and Total (in cell A6). Note that first-level headers cannot act across fences. In contrast, higher-level headers can act across fences.


Microsoft Excel - harvest.xls
File Edit View Insert Format Tools Data Window Help Adobe PDF
Arial
10
H10
Units
Clear
Sheet1
Ready
NUM

	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	17	14	47	
7						

Figure 10: 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. In the example spreadsheet shown in Figure 10, Fruit and Month cannot be assigned as first-level headers and belong to set U. 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. An n-level header cannot be assigned as a header for another header at level n.
2. A header at level n can only have one header at level $n+1$. If this rule were violated, the resulting dependent unit would not be well-formed.
3. If two cells are headers for a core cell, they cannot have a common header assigned to them. For example, in Figure 10, the core cell B4 has been assigned Apple (cell B2) and June (cell A4) as headers. If B2 and A4 were assigned a common header, B4 would result in a unit error (violation of Rule 3 in Section 2.1).

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.

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.)

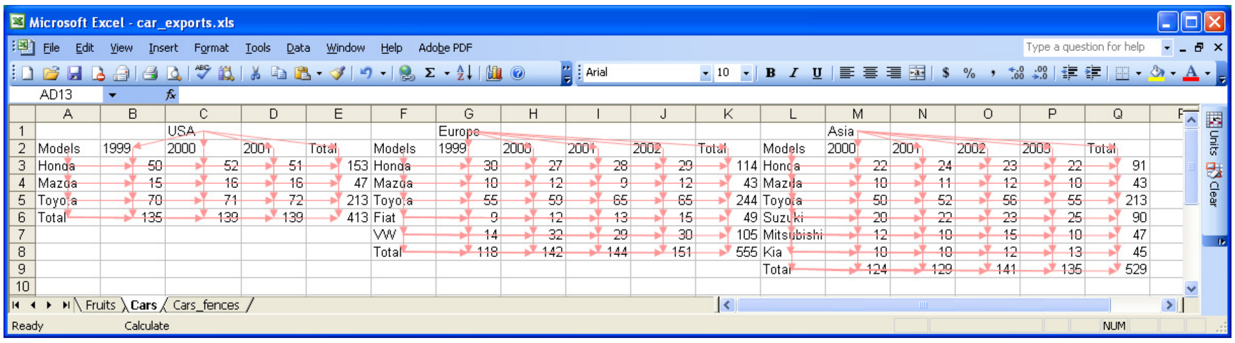
Elements of set U that do not satisfy these constraints are excluded from the set. The headers in set A are either row headers or column headers. Any element $u \in U$ can potentially be a higher-level column header for a subset of column headers $\{a_i | a_i \in A\}$ if the row number of u is less than 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 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 10 and then look at a more complicated case and discuss an extension to this algorithm.

In the example in Figure 10, Apple, Orange, Plum, and Total (in E2) are assigned as column headers and May, June, July, and Total (in A6) 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, A5, and A6 because of the spatial constraints discussed above. The cost for assigning Fruit as header for Apple, Orange, Plum, and Total is $0+1+2+3=6$. In contrast, the cost for assigning Month as the column header for Apple, Orange, Plum, and Total (in E2) is 10. Only one of these assignments (either Fruit or Month) can be selected. If the assignment for Month is selected, Fruit will remain unassigned, which 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, July and Total (in A6) is also 10. This assignment results in an overall cost of 16 since it does not conflict with the assignment of Fruit as column header. Therefore the systems assigns the headers as shown in Figure 10.

We consider a more complicated spreadsheet shown in Figure 11. The spreadsheet contains data for the number of cars exported to USA, 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 10 with Fruit), whereas others might prefer to position the higher-level header centered above its subheaders. In the current example,



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1			USA				Europe						Asia				
2	Models	1999	2000	2001	Total	Models	1999	2000	2001	2002	Total	Models	2000	2001	2002	2003	Total
3	Honda	50	52	51	153	Honda	30	27	28	29	114	Honda	22	24	23	22	91
4	Mazda	15	16	16	47	Mazda	10	12	9	12	43	Mazda	10	11	12	10	43
5	Toyota	70	71	72	213	Toyota	55	59	65	65	244	Toyota	50	52	56	55	213
6	Total	135	139	139	413	Fiat	9	12	13	15	49	Suzuki	20	22	23	25	90
7						VW	14	32	29	30	105	Mitsubishi	12	10	15	10	47
8						Total	118	142	144	151	555	Kia	10	10	12	13	45
9												Total	124	129	141	135	529
10																	

Figure 11: Car exports.

Europe and Asia have been positioned as per the first convention whereas USA 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 USA and Europe. Let this distance be d_1 (4 in this case). Starting at cell B2[1] we traverse distance d_1 to the right and reach cell F2. 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 USA. Similarly, we compute the column distance between Europe and Asia, say d_2 (6 in this case). We start at G2 and traverse distance d_2 to the right and reach cell M2. 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.

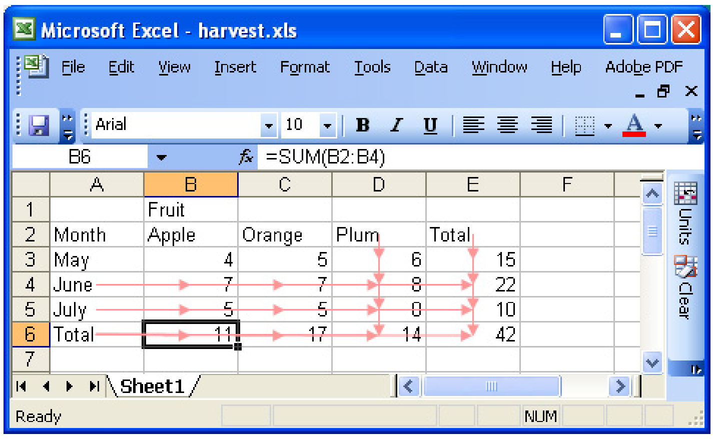
The resulting headers are as shown in Figure 11.

3.4 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 that contains a range error, the inferred headers are shown in Figure 12.

Since cell B2 participates in the aggregation formula in B5, the footer-to-core 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 12. 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

[1] Models in A2 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.


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 12: Unsatisfactory header inference.

the problem in this particular case, but it would be incorrect in general since the system would then always classify string values as headers.

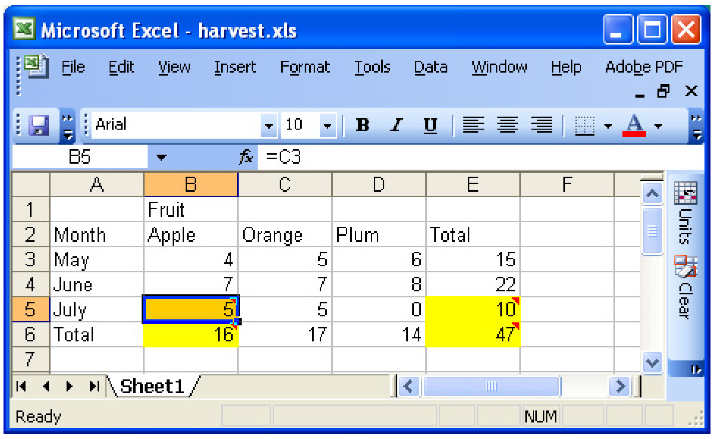
When the region-based classification algorithm is also enabled, it classifies the first row as a soft fence (one non-blank cell) and the seventh row and column F as hard fences. This results in all the cells in the second row being classified as headers with a confidence rating of 3, the cells in first column being classified as headers with a confidence rating of 5, the cells with formulas in row 6 and column E 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 10.

We have arrived at the set of classification algorithms described in Section 3.2 and their corresponding confidence ratings based on their performance over a wide variety of sample spreadsheets. The framework gives the developer the added flexibility of altering the set of classification algorithms used or their confidence ratings if the need arises.

UCheck works well in cases when headers are well organized into hierarchies and are placed in spatial proximity to the values they annotate. In fact, such a "labeling discipline" is partly assumed and exploited by the header inference. In practice we have observed that spreadsheets are, if at all, most of the time annotated in such a way. This fact is not too surprising because users introduce labels to make sense of the numbers in the spreadsheet. To have this effect labels must be placed in some structured and reasonable way. Consequently, strong deviation from this practice will cause UCheck to produce wrong results.

4 Error Detection using Units

The capability of the unit checker to detect errors arises from the fact that erroneous formulas often result in contradictory unit information. In the spreadsheet shown in Figure 13, cell B5 contains a reference to cell C3, which has the unit Fruit[Orange]&Month[May]. B5, by virtue of its row and column headers, has the unit Fruit[Apple]&Month[July]. Since B5 contains the reference to cell C3, the system applies rule $\text{REF}_{::}$ shown in Figure 5 and infers the unit for B5 as Fruit[Apple]&Month[July]&Fruit[Orange]&Month[May]. This unit cannot be simplified and is not 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 July have the common ancestor Month. Therefore, the system marks cell B5 to flag a unit error in this cell. Cells B6, E5, and E6 have formulas that have references to cell B5. Since the unit error propagates to these cells, they are marked in a lighter color.[2]



B5 =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	17	14	47	
7						

Figure 13: Identified reference errors.

This example demonstrates how the unit inference system can detect errors in a spreadsheet that result from wrong user input, such as overwriting a value by accidentally clicking in a wrong cell.

In the spreadsheet shown in Figure 14, the user has committed an error while entering the formula in B6—instead of finding the sum of cells B3 through B5, the formula tries to compute the sum of cells B2 through B4. Excel ignores this error and returns the result as 11 (the sum of the values in B3 and B4). Furthermore, this error propagates, resulting in an incorrect value in E6 as well since it has a reference to cell B6.

When we run the unit checker on this spreadsheet, the system infers the unit of cell B6 as Fruit|Month[May]&Fruit[Apple]|Month[June]&Fruit[Apple] since B2 has the unit Fruit, B3 has the unit Month[May]&Fruit[Apple], and B4 has the unit Month[June]&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, Month[May]&Fruit[Apple], and Month[June]&Fruit[Apple] do not have a common ancestor. So it is illegal to *or* them—violation of rule 4 discussed in Section 2.1. Since the aggregation formula in E6 refers to B6, the unit error propagates and causes E6 to be marked as an error.

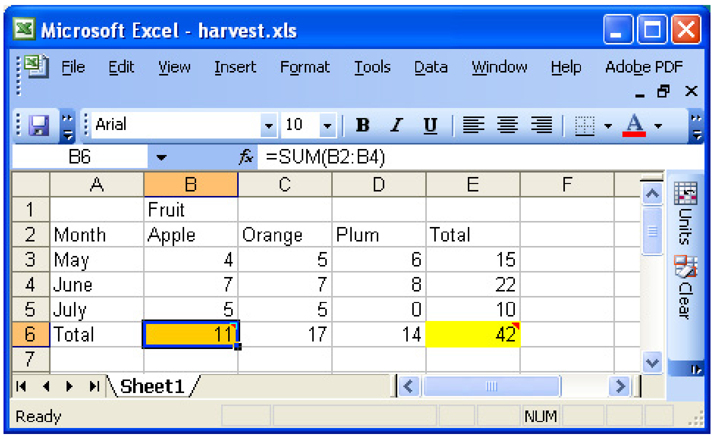
Again, the unit inference system can identify an error in the spreadsheet, this time a wrong range in a formula.

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 is sent to the backend server. The VBA system is shipped as an Excel add-in, see Figure 15.

The toolbar has two buttons as shown in the figures. When the user clicks the "Units" button, the unit checker is run. In case the sheet has unit errors, the system shades the cells that have unit errors. If the sheet does not have any unit errors, the system displays the message "Spreadsheet is free from unit errors" in the status bar at the bottom. This message disappears after the next change to the spreadsheet. The "Clear" button removes the unit-error shading from the spreadsheet.

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


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 14: Identified range error.

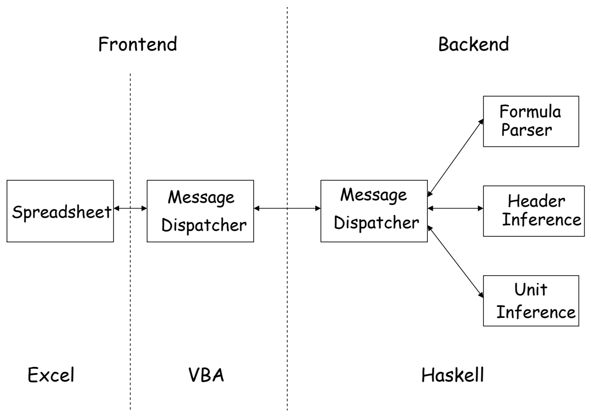


Figure 15: System architecture.

In the *debug mode*, the interface has a “Headers” button in addition to the “Unit” and “Clear” buttons. 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 10. We have enabled this representation for testing purposes so that we can verify the accuracy of the header inference system. 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 [17] on spreadsheets in science and engineering. The second set (set B) consisted of 18 spreadsheets developed by undergraduate Computer Science students as part of an introductory course on programming. From among all the sheets developed by the students as part of the coursework, we selected spreadsheets from each assignment, and also included those that had unusual spatial layout.

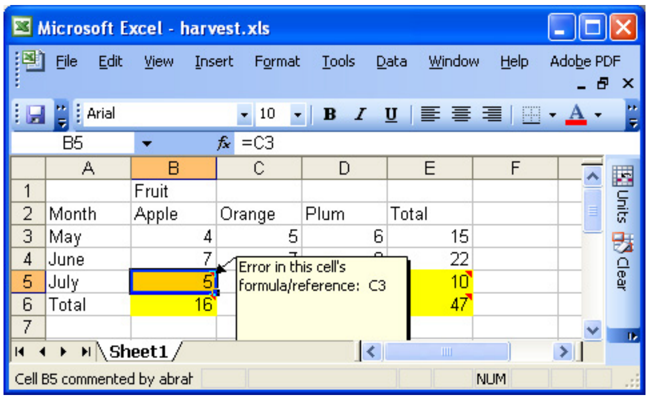
The header inference system incorrectly identified higher-level headers in some cases, especially with the sheets in set B. For example, in a couple of cases, the system identified the student's name and identification number as master column headers for the spreadsheet cells. Since the output from the 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 [4]. 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.

It is important to note that a generally accepted "representative" set of spreadsheets is not available. For the evaluation of our system, we consider spreadsheets in set A "expert" spreadsheets since they have been published with a book, whereas spreadsheets in set B are considered "novice" spreadsheets since they have been developed by students in an introductory course on programming. We also carried out an evaluation of the combined capabilities of UCheck and WYSIWYT by seeding errors in spreadsheets by using spreadsheet mutation operators [2]. The evaluation has shown that UCheck is very effective at detecting systematically-seeded spreadsheet errors as well [23].

As part of the work being done by the EUSES consortium [15] to help end users develop safer and more effective software, we conducted a continuing education course for high school teachers through Oregon State University. During a two week session, the teachers were introduced to auditing features available within Excel, the WYSIWYT testing methodology [32], the Gencel system [12, 13], and the UCheck system [1]. As part of the course, the teachers then used the systems to teach the importance of error prevention to a group of high school students. We


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

B5 =C3

	A	B	C	D	E	F
1		Fruit				
2	Month	Apple	Orange	Plum	Total	
3	May	4	5	6	15	
4	June	7	[illegible]	[illegible]	22	
5	July	5	[illegible]	[illegible]	10	
6	Total	16	[illegible]	[illegible]	47	
7						

Error in this cell's formula/reference: C3

Cell B5 commented by abrat

Figure 16: Fault localization feedback.

gathered feedback from the teachers and the students regarding their experiences and impressions about the UCheck system. Both the students and the teachers appreciated the fact that UCheck is fully automatic. The version of the system they were using did not provide detailed feedback on the cause of the errors or how to correct them. It used cell shading to communicate fault localization information, and also listed the references from a cell as shown in Figure 16. The information about references is important since unit errors can be caused only by incorrect references—constant values in a cell do not cause unit errors. Based on the feedback collected from the students and teachers, we are working on improving the visual feedback provided by UCheck as discussed in Section 8.

7 Related Work

The pervasiveness of errors in spreadsheets has motivated research into spreadsheet design [22, 41, 31], code inspection [25], quality control [29], testing [32], and consistency checking [14, 10, 9, 4]. Recently, there has also been work on improving the programming capabilities of spreadsheets [27]. This approach follows the guidelines offered by the Cognitive Dimensions of Notations [7] and the Attention Investment model [6].

The spreadsheet language Forms/3 [8] has been used to explore different approaches to developing error-free spreadsheets. The "What You See Is What You Test" (WYSIWYT) methodology for testing spreadsheets was developed within this framework [32]. "Help Me Test" (HMT) is a component of Forms/3 that automatically generates test cases for end users to test their spreadsheets [18]. Fault localization techniques to help with debugging of the spreadsheets have also been incorporated into the Forms/3 system [28, 33].

The use of assertions to identify erroneous formulas is presented in [9]. 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 [5] 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 [4] is most closely related to ours (since it builds on our original work in [14]). Their system supports two kinds of relationships between headers—*is-a* relationships link instances and subcategories, whereas *has-a* relationships describe 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 [4] 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 [4] do not use this constraint and allow the generalization of either *or* unit to fruits and thus also allow the illegal comparison.

A different approach to more reliable spreadsheets is to try to prevent errors in spreadsheets by allowing only those spreadsheet updates that are logically and technically correct that captures the essential structure of a spreadsheet and all of its future versions. We have developed a visual notation (called ViTSL for Visual Template Specification Language) to describe spreadsheet templates that capture the essential structure of a spreadsheet and all of its future versions. The user can develop spreadsheet templates using a visual interface to ViTSL [3]. A template can then be imported into Gencel [12, 13] which translates it into an initial spreadsheet instance, together with customized update operations for insertion/deletion of rows and columns. The advantage of this approach is that any spreadsheet created using the customized update operations, starting from a type-correct template, is proved to be free from omission, reference, or type errors.

8 Future Work

Although the present UCheck system works quite well in practice, there are several different areas in which improvements can be made. First, we can add more analysis algorithms to further improve the accuracy of cell classification. In this context, all kinds of formatting information provide valuable sources of input since formatting 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 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. To speed up the inference process, we are also looking at ways to support incremental header/unit inference. However, giving the user immediate feedback after every action might be too intrusive. A constantly running unit monitor requires 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”.

In the current implementation, the background of cells with unit errors is shaded as shown in Figure 13. The shading reflects the fact that a cell can be assigned an invalid unit in two ways.

1. Primary unit error: The unit of the cell under consideration is itself incorrect. In this case, the cell is shaded orange.

2. Propagation unit error: The cell contains a unit error because it references a cell containing a unit error. These cells are shaded yellow.

The users should direct their debugging efforts to the cells that have been shaded orange. For example, in the sheet shown in Figure 16, correcting the reference to cell C3 in cell B5, would remove all the unit errors. We plan to extend the error-feedback mechanism to include *context-specific* examples and *change suggestions*. Context-specific examples will be generated on-the-fly from the spreadsheet the user is working on. In the case of the error shown in Figure 16 we could use the column C (or column D) as a concrete example for the user to follow to correct the reference error in cell B5. We believe that this would go a long way in helping the users understand the errors and take corrective actions.

We are also working on generating suggestions for changes the user can apply to the spreadsheet to correct the errors. In the sheet shown in Figure 16, the unit for cell B5 cannot be reduced to a valid unit. We can use this unit error to generate suggestions for changes that would result in cell B5 having a valid unit. More than one suggestion would be possible in general and we can use heuristics to rank them from reasonable to unlikely, so the users can pick the one they deem fit. In the current example, three sample change suggestions that would result in cell B5 having a valid unit are given below.

1. Replace reference to cell C3 in B5 by a numerical constant.

2. Change header Apple in cell B2 to Orange and July in cell A5 to May.

3. Change header Orange in cell C2 to Apple and May in A3 to July.

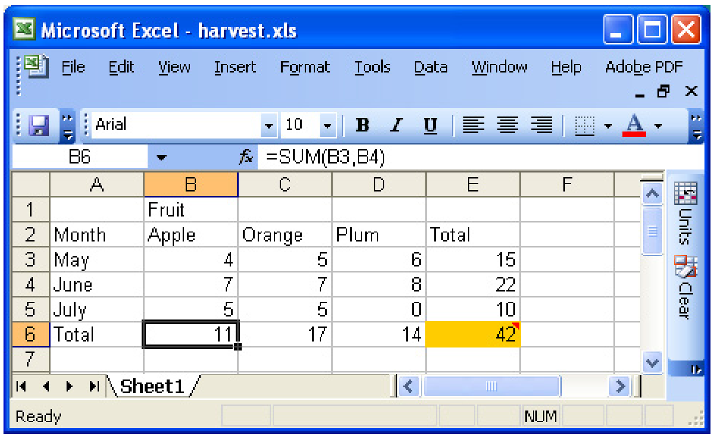
The first suggestion simply reduces the unit of B5 to Fruit[Apples]&Month[July] by getting rid of the reference. The second suggestion would result in the unit of B5 reducing to Fruit[Oranges]&Month[May] both by virtue of reference and position. Similarly, the third suggestion would result in the unit of B5 reducing to Fruit[Apples]&Month[July]. The first suggestion would have a higher rating because it is the *simplest* in terms of changes to be done to the sheet. Moreover, the second and third suggestions could be considered unlikely since they involve changes to the header information, and introduce duplicate headers.

In the spreadsheet shown in Figure 17, the formula in cell B6 does not include B5. The *incomplete* range in the formula is not reported as a unit error since cell B6 has the valid unit Fruit[Apple]&Month[May|June]. However, this error surfaces when we try to combine the value in B6 with those in C6 and D6 in the formula in cell E6. C6 and D6 have the units Fruit[Orange]&Month[May|June|July] and Fruit[Plum]&Month[May|June|July], respectively, and these cannot be combined with the unit of B6. This results in the error being reported in cell E6.

From a user interaction point of view, this is not ideal since the error gets reported away from its source. The change-suggestion mechanism discussed earlier in this section would be able to overcome this problem.

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 that 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.


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

B6 =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						

Figure 17: Incomplete range error.

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. Mutation Testing of Spreadsheets. 2006. Submitted for publication.

[3] 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.

[4] Y. Ahmad, T. Antoniu, S. Goldwater, and S. Krishnamurthi. A Type System for Statically Detecting Spreadsheet Errors. In *18th IEEE Int. Conf. on Automated Software Engineering*, pages 174–183, 2003.

[5] T. Antoniu, P. A. Steckler, S. Krishnamurthi, E. Neuwirth, and M. Felleisen. Validating the Unit Correctness of Spreadsheet Programs. In *26th IEEE Int. Conf. on Software Engineering*, pages 439–448, 2004.

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

[7] 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.

[8] M. M. Burnett, J. Atwood, R. Djang, H. Gottfried, J. Reichwein, and S. Yang. Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm. *Journal of Functional Programming*, 11(2):155–206, 2001.

[9] M. M. Burnett, C. Cook, J. Summet, G. Rothermel, and C. Wallace. End-User Software Engineering with Assertions. In *25th IEEE Int. Conf. on Software Engineering*, pages 93–103, 2003.

[10] 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.

[11] D. Cullen. Excel snafu costs firm $24M. *The Register*, June 2003.

[12] 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.

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

[14] 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.

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

[16] EuSpRIG. European Spreadsheet Risks Interest Group. `http://www.eusprig.org/`.

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

[18] M. Fisher II, M. Cao, G. Rothermel, C. Cook, and M. M. Burnett. Automated Test Case Generation for Spreadsheets. In *24th IEEE Int. Conf. on Software Engineering*, pages 141–151, 2002.

[19] J. Gallob. LinCom nears Budget Resolution. *Newport News Times*, May 2004.

[20] A. Givens. Brookhaven Corruption Probe, Investigators Made Mistakes, March 2004. `Newsday.com`.

[21] K. Godfrey. Computing Error at Fidelity's Magellan Fund. In *Forum on Risks to the Public in Computers and Related Systems*, January 1995.

[22] 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.

[23] J. Lawrance, R. Abraham, M. Burnett, and M. Erwig. Sharing Reasoning About Faults in Spreadsheets: An Empirical Study. In *IEEE Int. Symp. on Visual Languages and Human-Centric Computing*, 2006. To appear.

[24] Solution Matrix Ltd. We Just Lost Our Negotiating Elbow Room: The Pitfalls of Excel. *Cost/Benefit Newsletter*, June 2004.

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

[26] R. R. Panko. Spreadsheet Errors: What We Know. What We Think We Can Do. In *Symp. of the European Spreadsheet Risks Interest Group (EuSpRIG)*, 2000.

[27] 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.

[28] 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. In *IEEE Int. Symp. on Human-Centric Computing Languages and Environments*, pages 203–210, 2003.

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

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

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

[32] 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.

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

[34] C. Scaffidi, M. Shaw, and B. Myers. Estimating the Numbers of End Users and End User Programmers. In *IEEE Symp. on Visual Languages and Human-Centric Computing*, pages 207–214, 2005.

[35] A. Scott. Shurgard Stock Dives After Auditor Quits Over Company's Accounting. *The Seattle Times*, November 2003.

[36] R. E. Smith. University of Toledo loses $2.4M in projected revenue. *Toledo Blade*, May 2004.

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

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

[39] U.S. Department of Education. Audit of the Colorado Student Loan Program's Establishment and Use of Federal and Operating Funds for the Federal Family Education Loan Program, July 2003. Report ED-OIG/A07-C0009.

[40] U.S. Department of Health and Human Services. Review of Medicare Bad Debts at Pitt County Memorial Hospital, January 2003. Report A-04-02-02016.

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