Using Objects of Measurement to Detect Spreadsheet Errors

Michael J. Coblenz

July 2005
CMU-CS-05-150
CMU-HCII-05-102

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the Senior Honors Thesis program in the School of Computer Science at Carnegie Mellon University

This research was made possible by the guidance of Andrew J. Ko, Brad A. Myers, and Frank Pfenning.

Abstract

There are many common errors in spreadsheets that traditional spreadsheet systems do not help users find. This paper presents a statically-typed spreadsheet language that adds additional information about the objects that spreadsheet values represent. By annotating values with both units and labels, users denote both the system of measurement in which the values are expressed as well as the properties of the objects to which the values refer. This information is used during computation to detect some invalid computations and allow users to identify properties of resulting values.

This research was partially supported by the EUSES consortium under NSF ITR CCR-0324770. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect those of the National Science Foundation.

Keywords: spreadsheets, spreadsheet errors, unit systems, spreadsheet languages

1. Introduction

Spreadsheets are used both by home users for small calculations as well as by business users for mission-critical applications involving millions of dollars. Unfortunately, errors in spreadsheets are as ubiquitous as spreadsheets themselves, with 20% to 40% of spreadsheets containing errors [8].

Some recent work has focused on software engineering techniques that may be applied to the spreadsheet development process. For example, Rajalingham et al. describes improvements in the process itself [9], and Rothermel et al. describes tools to help test and debug spreadsheets [10]. These are useful additions to an already successful language paradigm, but they do not attempt to improve the language itself. Work on improving spreadsheet languages has focused primarily on augmenting them with units. XeLda [3] allows users to define their own units, and propagates them through computations. Apples and Oranges [6] defines a somewhat different form of unit, based on inferences from headers in tables of spreadsheets.

While these approaches can help users detect errors in values and units, errors based on the object being measured can go unnoticed. This paper introduces a new spreadsheet system called SLATE ("A Spreadsheet Language for Accentuating Type Errors"), which separates the unit from the object of measurement, and defines new semantics for spreadsheets so that both the unit and the object of measurement are taken into consideration. Unlike the standard semantics for units, the semantics of operations on objects of measurement are not obvious; it is necessary to choose an intuitive approach for propagating information through calculations. By redefining the semantics of traditional spreadsheet operations, such as addition and multiplication, the system can generate additional information about results that reveals formula errors.

For example, a user might mistakenly multiply pounds of apples by the price per pound of oranges. A traditional spreadsheet showing only values would hide this error by displaying only the result, in dollars. Even considering units would not reveal this error. SLATE reveals the problem by showing that the result has properties of both apples and oranges:

```
10 lb. (apples) * $0.50 / lb. (oranges) = $5 (apples, oranges).
```

The next section briefly discusses related work. Afterward is an example of an error that SLATE would help the user detect. Then, core concepts of the language are introduced, and in the fifth section is a

detailed description of the operators of the language and justifications for their design. This is followed by a discussion of user interface issues for this language. The conclusion discusses future steps for SLATE.

2. Related Work

Like SLATE, other systems have had the goal of improving on the spreadsheet paradigm. Forms/3 [4] takes a functional programming perspective, and extends the full functional programming paradigm to a spreadsheet context. Forms/3 avoids the requirement of rows and columns, instead allowing any configuration of cells. This philosophy was adopted in the design for SLATE: although the examples here are in a standard table layout, the language itself would be suitable for another visual arrangement of cells.

Apples and Oranges [6] is the most closely related work. In it, Erwig and Burnett develop a unit system whereby the system infers units for cells using a header cell inference algorithm [1]. The system defines the spreadsheet operations in terms of its unit system, and flags cells if its inference algorithm suggests an error. However, the inference algorithm is opaque, and if users do not format the spreadsheet as the authors intended, units may be inferred incorrectly (although Burnett and Erwig suggest in [5] ways in which users may customize the inference process). Furthermore, because the units can become very complicated, they are not suitable for display to users. Units are also limited to header data; no other data can be used.

Another spreadsheet error detection system is described in [2], in which Ahmad et al. describe a system that identifies header cells for each cell and uses *is-a* and *has-a* relationships between cells to give units to values. Like Erwig and Burnett's work, it has the goal of automatically detecting errors and highlighting cells that contain them; thus, the inference algorithm is opaque, and headers must be either manually chosen or potentially inferred incorrectly.

Kennedy describes an ML-style functional programming language that includes dimensions [7]. Although it is not presented in a spreadsheet context, it is groundwork for statically-typed languages that include dimensions. Like other systems that include only units or dimensions, it cannot detect errors in objects of measurement.

3. An Example: Orchard Records

Apples (per lb.)	Oranges (per lb.)
$0.45	$0.50

Fruit	Fruit Sold (lbs.)	Revenue
Apples	312	$140.40
Oranges	399	$179.55

Figure 1. An incorrect calculation in a spreadsheet.

To illustrate how SLATE reveals errors, consider Figure 1, where a user attempted to calculate revenues for two types of fruit: apples and oranges. Instead of multiplying each weight of fruit by the corresponding cost, the user accidentally multiplied each weight by the cost per pound of apples. Conventional spreadsheets only display the result of the calculation, so the source of the error is not visible. Spreadsheets that consider only units would not reveal this error either, since both values under consideration have the same units: $ / lb. Because of the particular values in the cells, the user is unlikely to find this error by estimating the correct result and comparing to the computed values. In fact, the mistake has been completely hidden, only to be found by a careful inspection of the formulas.

SLATE reveals these errors by displaying additional information in the cells: in addition to displaying a unit, it displays a *label*, which is a list of attributes pertaining to the value in the cell. To visually separate labels from units, they are enclosed in parentheses when displayed.

In Figure 2, the same calculation from Figure 1 is performed in SLATE. In the "Revenue" column, the amounts are treated as measurements of fruit. The first row measures the cost of apples. The second row, however, appears to measure the cost of fruit that is simultaneously apples and oranges. This is obviously wrong; the user expected the cell to have only the attributes of oranges, since the calculation has nothing to do with apples. By computing and displaying these labels based on the labels the user entered for the original information, the system can help users detect otherwise hidden errors.

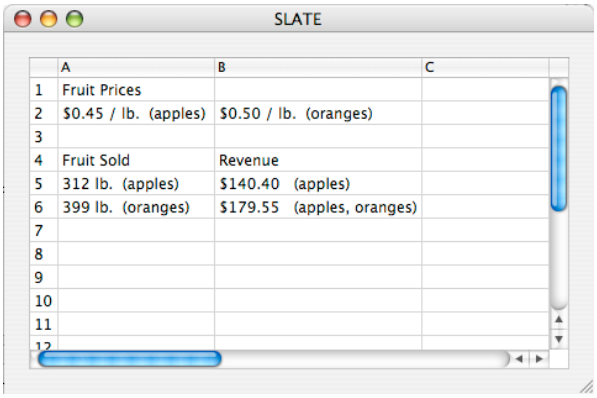

Figure 2. The spreadsheet from Figure 1, using SLATE. The revenue for oranges is incorrect. SLATE computed the contents of the Revenue cells, including the labels.

4. Language Introduction

This section discusses the additional data that SLATE must maintain to reason about units and labels.

4.1. Values, Units, and Labels

In SLATE, every expression has three attributes: a value, a unit, and a label. The value is the same as spreadsheets would normally contain. Units, such as meters, kilograms, and seconds, indicate the way in which a measurement was made. They capture information about the scale at which the measurement was taken and the dimensions of the measurement (although this system does not treat dimensions, such as weight, separately from the units, such as pounds, as discussed in Kennedy [7]). SLATE adds *labels*, which define characteristics of the objects of measurement. For example, a cell referring to 25 pounds of apples might read “25 lbs. (apples)”. In this example, the label is “(apples)”. A cell referring to apples picked in September might have the label “(apples, September)” because the value in the cell has characteristics of both apples and September.

4.2. Contexts

Since SLATE must understand arbitrary objects being measured, it must be customizable to work for many different kinds of objects. For example, a construction company expects a spreadsheet to understand plywood and concrete; a farm expects it to understand fruit, vegetables, and fertilizer. An interior decorator would like “orange” to refer to a color, whereas an orchard manager would like “orange” to refer to a kind of fruit. These are different, since they have different subtypes: the color might have subtypes of “dark orange” and “red-orange,” but the fruit might have subtypes of “Navel” and “Valencia.”

To accomplish this, each spreadsheet refers to two contexts: a unit context and a label context. The unit context defines the base units that are available to the user. Base units are the primitive units that, when multiplied, form other units; thus, units in spreadsheets are formed from these base units. The “quantity” unit is a special case of a base unit, and is used for referring to quantities when counting discrete objects, such as “4 quantity (apples)”. It may be abbreviated “qty.”

The label context forms the core of this project. The structure of the label context reflects the observation that many real-world concepts and objects are hierarchical. Operations are defined on the

labels so that generalizations up the hierarchy take place where appropriate. Therefore, the label context is a tree, where each node is a particular concept. There is an edge from node n_1 to node n_2 if objects of type n_2 have all of the properties of objects of type n_1. For example, in Figure 3, there is an edge from "Apples" to "Red Delicious" because red delicious is a kind of apple. Red Delicious apples have the properties of apples, and also some additional properties that not all apples share.

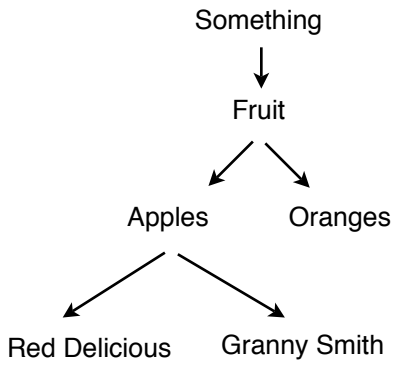

Figure 3. A small label context.

Alternatively, one might define contexts to be directed acyclic graphs rather than trees. Although trees may result in less compact representations for certain kinds of objects, they have the advantage of ensuring clear semantics of the operations defined in section 5.

The example label context in Figure 3 might be suitable for a small orchard. The `Something` node represents the most general type of object; it is named as such to warn the user of a potentially dangerous generalization.

5. Language Specification

5.1. Units

A unit context, denoted by Γ, is a set of base units. A base unit is used as in Kennedy's work [7]; it can be thought of as a unit which cannot be expressed in simplest form as a product of other units.

A unit is a product of integer powers of base units. Let B range over elements of Γ. Then units μ are defined as follows, with $n \in \mathbb{Z}$ (i.e. n is an integer):

$\mu ::= 1 \mid \mu \cdot B^n$

1 is the identity unit; it is never displayed to users.

Each unit has a canonical representation. In particular, the base units are sorted lexicographically (there may be at most one base unit with a given name). There may be no more than one appearance of a particular base unit in the canonical representation; multiple appearances are combined by summing the exponents. Units are considered to be equivalent if they have identical canonical representations. The symbol = will be used to represent equivalences.

5.2. Labels

A label context Λ is a tree, comprised of concepts (C) and edges (E), where there is a path from the root (called `Something`) to each other node. Thus, $\Lambda = (C, E)$. Each node in the graph represents a concept. $c \in \Lambda$ will serve as an abbreviation for $c \in C$.

A label λ is a set of nodes from the set C, such that there is no path in Λ between any two nodes in λ. It represents objects that have the properties of all of its nodes. For example, the set {apple, ripe} is a label that represents the set of objects that are apples and are also ripe. Of course, this label is only defined in an appropriate context; {apple, ripe} is not a valid label otherwise. The empty label, {}, represents the `Something` node. The relation *descendent* (c_1, c_2) holds if and only if there is a path in Λ from c_2 to c_1.

The relation $\leq_\ell$ between pairs of labels is defined as follows:

$$\lambda_1 \leq_\ell \lambda_2 \Leftrightarrow \forall\, c_2 \in \lambda_2\, .\, \exists\, c_1 \in \lambda_1.\ \textit{descendent}\ (c_1, c_2)$$

Claim: $\leq_\ell$ is a partial ordering on labels.

Proof: Reflexivity: $\forall c$. *descendent* (c, c), since $\forall c \in \lambda$, there is a trivial path from *c* to *c*.

Antisymmetry: Suppose $\lambda_1 \leq_\ell \lambda_2$ and $\lambda_2 \leq_\ell \lambda_1$. The fact that $\lambda_1 = \lambda_2$ follows directly from antisymmetry of the descendent relation on trees: let $c_1 \in \lambda_1$ and $c_2 \in \lambda_2$. We have *descendent* (c_1, c_2) and *descendent* (c_2, c_1), so $c_1 = c_2$.

Transitivity: Suppose $\lambda_1 \leq_\ell \lambda_2$ and $\lambda_2 \leq_\ell \lambda_3$. Let $c_3 \in \lambda_3$ be given. By the definition of $\leq_\ell$, $\exists$ $c_2 \in \lambda_2$ such that *descendent* (c_2, c_3). Likewise, $\exists\ c_1 \in \lambda_1$ such that *descendent* (c_1, c_2). But elementary properties of graphs show that the descendent relation is transitive, so we have *descendent* (c_1, c_3), as required. □

5.3. Types

Together, the unit and label form a type: $\tau = (\mu, \lambda)$. The labels impose a subtyping relation $\leq$, defined as follows:

$$(\mu_1, \lambda_1) \leq (\mu_2, \lambda_2) \Leftrightarrow \mu_1 = \mu_2 \text{ and } \lambda_1 \leq_\ell \lambda_2$$

5.4. Abstract Syntax for Expressions

Let f represent a floating-point value, and let ε be the empty expression. References to cells will be represented by *ref*; *range-ref* represents a reference to a set of cells. SLATE's simple spreadsheet language defines expressions *e* as follows:

e ::= ε | f μ λ | e + e | e - e | e / e | e * e | *ref*

| MAX (*range-ref*) | MIN (*range-ref*)

| AVG (*range-ref)*

| *string*

A future version of this work should include boolean values and additional functions; they are not included here for simplicity.

5.5. Addition and Subtraction

Expressions (other than those that have errors in evaluation) may be added or subtracted if and only if they have equivalent units. The restriction to permit adding or subtracting only expressions with equivalent units maintains the standard interpretation of units: because units express the measurement system, permitting these operations for values of different units would result in nonsense.

Errors are propagated, so that if an operand has an error in evaluation, so does the result. Values are simply added or subtracted; no conversions are performed.

To determine the label of the result of an addition or subtraction operation, SLATE derives the least general label that includes all of the properties in both of the operands.

Let *intersection-with-paths* be a function from label pairs to labels, which when given a pair of labels (λ_1, λ_2) returns the set of nodes:

$$\{n \mid (n \in \lambda_1 \text{ and } \exists n' \in \lambda_2\text{: descendent } (n', n)) \text{ or } (n \in \lambda_2 \text{ and } \exists n' \in \lambda_1\text{: descendent } (n', n))\}$$

Let *parents* be a function from labels to labels, which when given a label λ, returns the union of the sets of parents of the nodes in λ for the given context.

The label derived for addition and subtraction is as follows:

```
fun add-sub-labels (λ1, λ2) =
  if λ1 = {} then {}
  else if λ2 = {} then {}
  else
    let intersection =
     intersection-with-paths (λ1, λ2)
    in
     intersection ∪
       add-sub-labels (parents (λ1\intersection),
                       parents (λ2\intersection))
   end
```

This function defines a unique label, given λ_1, λ_2, and the context Λ, since it is deterministic, and terminates (the tree is finite).

For example, using the context in Figure 3:

```
5 lbs. (apples) + 2 lbs. (oranges) = 7 lbs. (fruit)
```

because apples and oranges are both fruit. Similarly, in a context where September has been defined:

```
5 lbs. (apples, September) + 2 lbs. (oranges, September) = 7 lbs (fruit, September).
```

Notice how apples and oranges are combined into fruit, but because September is in both labels, it is copied into the final label. Likewise:

```
5 lbs. (apples) + 2 lbs. (oranges, September) =  7 lbs. (fruit)
```

in a context where the parent of September is `Something`. Figure 4 shows how `add-sub-labels` computes the resulting label in this example.

Iteration	λ_1	λ_2	`intersection`	Result
0	{apples}	{oranges, September}	{}	{} ∪ {fruit} = {fruit}
1	{fruit}	{fruit}	{fruit}	{fruit} ∪ {}
2	{}	{}	*N/A*	{}

Figure 4. Computation of the result label:
`5 lbs. (apples) + 2 lbs. (oranges, September) =  7 lbs. (fruit)`

The goal of this choice of definition of addition and subtraction is for the labels to succinctly express the properties of the resulting object. Addition can be thought of as a union of sets of objects. The label of the result expresses the properties that are guaranteed of an arbitrary element of the result. The only properties that can be guaranteed are those that are shared among all the objects in the set.

It might seem, on the surface, that subtraction could be defined in a manner opposite to addition, since the two operations are inverses: when evaluating e_1 - e_2, subtraction would keep the label of e_1, but would be illegal if e_2 were not a subtype of e_1. This is attractive because it would seem to preserve the semantics of labels referring to sets. However, this approach would be inconsistent because it would cause the label of the result of an addition operation to depend on the signs of the values. This instability would be especially confusing for a user, who could find that small changes in values in one part of a spreadsheet caused errors in a dependent part of the spreadsheet due to sign changes in dependent values.

Another possible choice of definition for addition and subtraction would be to require equality in the labels, in addition to equivalence in the units. This would be overly restrictive, however, since users frequently need to compute sums of different kinds of objects. For example, for the purpose of finding the total weight of a shipment consisting of many different items, the fact that the items are different is irrelevant; the user only wants the total weight. SLATE expresses this by choosing the most general label (perhaps `Something`). The user is therefore alerted to the fact that the items are different. If this is an error, the fact that the label unexpectedly became more general serves as a warning to the user.

5.6. Multiplication and Division

Multiplication and division of two non-error values is always legal; the resulting unit is the product of the units of the operands. Errors are defined to propagate, so that if an operand has an error in evaluation, so does the result. For labels, multiplication and division are defined so that the result label includes all of the properties of the operands' labels. Note that labels are never in the denominator of any expression, unlike the behavior of units: when two values are divided, the divisor's *unit* is in the denominator of the result, but the divisor's *label* is not. Instead, a new label is computed as per `mult-div-labels` below and attached to the resulting value.

It would be redundant for a label to contain both a node and one of its ancestors; in these cases, the ancestor is discarded. Assume that the function *eliminate-ancestors*, when given a label λ, returns a new

label λ′ that is the same as λ, but the ancestors of ancestor-descendent pairs in λ have been removed. Multiplication and division of labels is implemented as follows:

```
fun mult-div-labels (λ1, λ2) =
  eliminate-ancestors (λ1 ∪ λ2)
```

For example:

```
2 qty. (apples) * $0.50 / qty. (apples) = $1.00 (apples)
```

But:

```
2 qty. (apples) * $0.50 / qty. (oranges) = $1.00 (apples, oranges)
```

Also, in the following example, `fruit` is an ancestor of `apples`, so it is not included in the result:

```
2 qty. (apples) * $0.50 / qty. (fruit) = $1.00 (apples)
```

The label (apples, oranges) is different both syntactically and semantically from the label (fruit). The label (apples, oranges) denotes objects that have all the properties of apples and also all the properties of oranges; the user will observe that not only was one of apples or oranges not expected in the result, but that there are no objects like this. The label (fruit) would denote objects that have the properties of fruit, without specifying what other properties of apples or oranges they might have.

The choice of definitions for multiplication and division represents a tradeoff between recording the history of the computation—resulting in large, unreadable labels—or erasing it, and hiding potential errors. One concern with the definition chosen here is that it may maintain too much of the computation history. For example, the product of a list of different items would result in a very large label, since SLATE would take the union of the elements of the operands' labels. However, spreadsheets are not commonly used to compute products of large sets of items; although this is a potential use (for example, in calculating geometric means), SLATE optimizes for the common case: users sum large lists, but tend to multiply only small lists. Furthermore, in most spreadsheets, the chain of computation may be relatively short. If this is the case, the label will probably not become too large.

An alternative to the union definition chosen for SLATE, arising from the concern above, is to use intersections of the sets, rather than unions. The intersection operator would be defined similarly to the algorithm used for labels in addition and subtraction. However, although this approach would result in smaller labels, multiplication does not correspond to the item-property semantics described for addition

and subtraction (wherein the result of the operation describes properties of objects in the union of the two sets), so its interpretation would be unclear: multiplication in this context is not simply repeated addition, as shown by the behavior of units with multiplication. Furthermore, this alternative definition would hide errors: rather than indicating errors by displaying unexpected attributes, it would require users to notice the absence of expected attributes.

A natural question is whether this definition should also be used for addition and subtraction. However, the observation that addition is commonly used for lists of dissimilar items precludes this choice, since it would frequently result in very long, unreadable labels. Further, it would break the semantics of labels as expected for addition: items in the result set would have only some of the properties of the label attached to the result, rather than all of them. This property is less important for multiplication and division, however; users familiar with operations with units know that units behave differently for multiplication and division than for addition and subtraction.

Another possible choice would be to define labels so that multiplication and division could be performed in a manner similar to those operations on units. In particular, there would be no simplification between labels with multiplication; the system would generate exponents if labels were identical, and cancel identical labels on multiplication if their exponents summed to zero. Division would be defined as the inverse of multiplication. This definition, despite its intuition and appeal, is not an appropriate choice for this system. Because of the large variety of possible labels, cancellation would occur much less frequently than it would with units. The result is that labels would become very complicated, and therefore unreadable. For example, under this interpretation:

```
10 lbs. (red delicious) * $.50 / (lb. (apples)) = $5 lbs. (red delicious) / (lb. (apples))
```

This calculation might be done to compute the cost of 10 pounds of red delicious apples in a context where all apples cost $.50 per pound. No cancellation occurs in this example, resulting in a complicated, unreadable label with unclear semantics. Users would likely choose to ignore the label rather than examining it. Furthermore, the correct assignment of labels to the price per pound of apples is unclear: should the unit and label be `$.50 (apples) / lb.`, or `$.50 / (lb. (apples))`, or `$.50 (apples) / (lb. (apples))`?

One refinement attempt would be to consider labels to be contravariant in the denominator, meaning the system would cancel labels if the label in the denominator were a subtype of the label in the numerator. This would improve the previous example, resulting in $5. However, even in cases where

cancellation occurs, the system would lose important information as computation proceeded. The value $5 does not reflect the fact that it is a measurement of red delicious apples (or even, more generally, apples). This loss of information can hide errors caused by improper uses of this computed value. For example, it might be incorrectly added to $10 (oranges), under the mistaken assumption that the $5 was a measurement of oranges. This would result in $15, hiding the fact that this is in fact fruit. Users would begin to ignore the frequent disappearance of labels, causing them to also ignore potential errors.

5.7. Aggregate Operators

The aggregate operators defined in SLATE are MAX, MIN, and AVG. For all of these operators, the units of the operands must be equivalent (otherwise, comparisons or additions of the values would be meaningless).

The label of the result is as defined for addition and subtraction, extended to *n* labels instead of only two. For MAX and MIN, it would be tempting to instead copy the label from the maximum or minimum value into the result's label. However, the label of the result should depend only on the labels of the operands, rather than on their values. For example, suppose a user calculated the MAX of `3 qty. (apples)` and `5 qty. (oranges)`. This is legal, since both values have units of `qty.` However, the label of the result must be `(fruit)` rather than `(oranges)`. Otherwise, the label of the result would change if the number of apples changed to be larger than the number of oranges, resulting in changes propagating throughout the spreadsheet. This could be confusing to the user. Although there may be cases where the user might prefer this choice, type theory dictates that types should only depend on types of constants, not their particular numeric values.

5.8. Properties of Labels

It will be shown that labels form a lattice under the operations used for multiplication and division, and addition and subtraction. Let $\leq_\ell$ be the partial ordering on labels as given above. Let $\wedge$ be the operation defined for multiplication and division of labels, and $\vee$ be the operation defined for addition and subtraction of labels. Assume Λ is a label context. Define:

$\Delta = \{S \subseteq \Lambda \mid \nexists c_1, c_2 \in S$. descendent (c_1, c_2) or descendent $(c_2, c_1)\}$

Thus, Δ is the set of all labels for the given label context Λ. First, notice that the completion of Δ under the partial ordering $\leq_\ell$ is a lattice. Therefore, it remains to show that $\wedge$ is the infimum, i.e. the greatest lower bound, and $\vee$ is the supremum, i.e. the least upper bound, with respect to $\leq_\ell$.

Claim: $\lambda_1 \wedge \lambda_2 = \inf \{\lambda_1, \lambda_2\}$ with respect to $\leq_\ell$.

Proof: First, $\lambda_1 \wedge \lambda_2 =_{def}$ `eliminate-ancestors` $(\lambda_1 \cup \lambda_2)$.

1. $\lambda_1 \wedge \lambda_2 \leq_\ell \lambda_1$ and $\lambda_1 \wedge \lambda_2 \leq_\ell \lambda_2$

   Without loss of generality, let $c \in \lambda_1$ (the proof for $\lambda_1 \wedge \lambda_2 \leq_\ell \lambda_2$ is symmetric). If $c \in$ `eliminate-ancestors` $(\lambda_1 \cup \lambda_2)$, then it is proved. Otherwise, notice that the only elements of λ_1 that are not in $\lambda_1 \wedge \lambda_2$ have descendants in $\lambda_1 \wedge \lambda_2$. Therefore, there is a path from c to some element of $\lambda_1 \wedge \lambda_2$.

2. $\lambda_1 \wedge \lambda_2$ is the greatest of the lower bounds.

   Let λ such that $\lambda \leq_\ell \lambda_1$ and $\lambda \leq_\ell \lambda_2$. It remains to show that $\lambda \leq_\ell \lambda_1 \wedge \lambda_2$. Suppose instead $\lambda_1 \wedge \lambda_2 \leq_\ell \lambda$. Let $c \in \lambda$. There is a path from c to some element of `eliminate-ancestors` $(\lambda_1 \cup \lambda_2)$. Therefore, there is a path from c to some element c′ of $\lambda_1 \cup \lambda_2$. Since there is a path from c′ to c, and Λ is a tree, c = c′. So, $\lambda = \lambda_1 \wedge \lambda_2$.

Claim: $\lambda_1 \vee \lambda_2 = \sup \{\lambda_1, \lambda_2\}$ with respect to $\leq_\ell$.

Proof: Let $\lambda = \lambda_1 \vee \lambda_2$. The algorithm is below, for convenience:

```
fun add-sub-labels (λ1, λ2) =
  if λ1 = {} then {}
  else if λ2 = {} then {}
  else
    let intersection =
      intersection-with-paths (λ1, λ2)
    in
      intersection ∪ add-sub-labels  (parents  (λ1\intersection),
                                      parents (λ2\intersection))
  end
```

1. It must be shown that the given operation produces a valid label, i.e. there are no ancestor-descendent pairs in the result. This holds for the result of `intersection-with-paths` (λ_1, λ_2) by definition, and holds for the recursive call by strong induction on *min* (*depth* (λ_1), *depth* (λ_2)). Since the intersection was removed from each of λ_1 and λ_2, there can be no paths from elements of the result of the recursive call to elements of the intersection.

2. $\lambda_1 \leq_\ell \lambda_1 \vee \lambda_2$ and $\lambda_2 \leq_\ell \lambda_1 \vee \lambda_2$

Define *depth* (λ) to be the maximum path length from the root in Λ to any node of λ. *depth* ({}) is defined to be 0.

Proof by strong induction on *min* (*depth* (λ_1), *depth* (λ_2)).

Base case: suppose $\lambda_1 = \{\}$. But $\{\} \leq_\ell \{\}$, so it is proved for this case. The case where $\lambda_2 = \{\}$ is symmetric.

Induction step: Assume for all λ_1, λ_2 where *min* (*depth* (λ_1), *depth* (λ_2)) $\leq k$:

$\lambda_1 \leq_\ell$ `add-sub-labels` (λ_1, λ_2), and

$\lambda_2 \leq_\ell$ `add-sub-labels` (λ_1, λ_2).

Suppose *min* (*depth* (λ_1), *depth* (λ_2)) = k+1. Let:

I = `intersection-with-paths` (λ_1, λ_2), as defined above.

If $I \neq \varnothing$, then let $n \in I$ (otherwise, trivially, $\lambda_1 \leq_\ell I$ and $\lambda_2 \leq_\ell I$). By the definition of `intersection-with-paths`, there is a path from n to some element of each of λ_1 and λ_2. Therefore, $\lambda_1 \leq_\ell I$ and $\lambda_2 \leq_\ell I$.

Define:

P = `add-sub-labels (parents (`λ_1`\intersection), parents (`λ_2`\intersection))`

Notice that:

depth (`parents (`λ_1`\intersection)`) < *depth* (λ_1), and

depth (`parents (`λ_2`\intersection)`) < *depth* (λ_2).

Therefore, the induction hypothesis applies to P, so `parents (`λ_1`\intersection)` $\leq_\ell$ P and `parents (`λ_2`\intersection)` $\leq_\ell P$. It remains to show that $\lambda_1 \leq_\ell P \cup I$ and $\lambda_2 \leq_\ell P \cup I$. It

has already been shown that there is a path from each element of I to some element of each of λ_1 and λ_2, so it remains to show this for P. But there is a path from each element of P to a parent of some element of each of λ_1 and λ_2; by the definition of *parent*, we have the required result. Therefore, $\lambda_1 \leq_\ell \lambda_1 \vee \lambda_2$ and $\lambda_2 \leq_\ell \lambda_1 \vee \lambda_2$.

3. $(\lambda \leq_\ell \lambda_1 \vee \lambda_2)$ & $(\lambda_1 \leq_\ell \lambda)$ & $(\lambda_2 \leq_\ell \lambda) \Rightarrow \lambda = \lambda_1 \vee \lambda_2$

   Proof: Let λ such that $(\lambda \leq_\ell \lambda_1 \vee \lambda_2)$ & $(\lambda_1 \leq_\ell \lambda)$ & $(\lambda_2 \leq_\ell \lambda)$. If $\lambda = \varnothing$, $\lambda_1 \vee \lambda_2 = \varnothing$ and it is proved. Otherwise, by the definition of $\leq_\ell$, λ contains an element n for which there is an element $n' \in \lambda_1 \vee \lambda_2$ such that n is a descendent of n'. `Add-sub-labels` iterates toward the root of the tree. Therefore, since it included n in its result, it must have traversed n' also. If $n = n'$, then either there is a different pair n and n' with $n \neq n'$, or $\lambda = \lambda_1 \vee \lambda_2$ and it is proved; or `add-sub-labels` did not select n' for the result. According to the specification of `add-sub-labels`, it must be the case that $n' \notin$ `intersection-with-paths` (λ_1, λ_2). Therefore, either there is no element of λ_1 that is a descendent of n' (violating $\lambda_1 \leq_\ell \lambda$), or there is no element of λ_2 that is a descendent of n' (violating $\lambda_2 \leq_\ell \lambda$). Thus, it must be the case that $\lambda = \lambda_1 \vee \lambda_2$.

Therefore, $(\Delta, \wedge, \vee)$ is a lattice.

5.9. Asymptotic Efficiency of Label Operations

Since the label operations are *meet* (i.e. $\wedge$, the greatest lower bound) and *join* (i.e. $\vee$, the least upper bound) on a lattice, SLATE could take advantage of a standard implementation of a lattice.

For the implementation given above, meet consists of a single `eliminate-ancestors` operation with a union. A hash table could make this run in time linear in the number of nodes in the labels: a hash table could map from node pairs (n_1, n_2) to *true* if n_1 is an ancestor of n_2, and *false* otherwise.

Each stage in the iteration for the join operation requires an `intersection-with-paths` operation, which could run in time linear in the number of nodes in the labels. In a tree, the set of parents of a set of nodes is no larger than the original set; thus, if h is the height of the tree and n is the size of the

sets of nodes in the labels for which the join must be computed, the algorithm runs in O(*hn*) time if `intersection-with-paths` is linear in the number of nodes in each set.

6. User Interface Issues

6.1. Displaying Units and Labels

The system does not use labels to discover errors automatically; rather, they serve as additional information for users. This additional information can help users find errors by showing potentially unexpected properties of the results of the computations, as in the example in Figure 2. Therefore, the visual representation of labels is central in determining SLATE's effectiveness.

Despite the additional space required for units and labels, in many cases the additional information they provide can remove the need for extraneous annotations. For example, Figures 1 and 2 express the same data, but in the second table, Figure 2 uses one fewer column because the information can be expressed with labels. Of course, the benefit is less when there are many columns.

6.2. Editing Units and Labels

Another central factor that determines SLATE's effectiveness is the ease with which users may enter labels. If users choose not to enter labels, the system does not provide benefits.

One guiding principle that suggests some potential design ideas is *surprise-explain-reward* [11]. People are willing to go to some effort if they will be rewarded. For example, users may be willing to repair unit errors, since the system rewards them by evaluating their formulas. One possibility would be to give nonsense labels to values without labels. A tooltip would explain the system, and upon entering a valid label, the strange labels would disappear, potentially revealing downstream errors in the spreadsheet.

In the current implementation, users must type units and labels as text. This has the advantage of permitting users to enter data with only a keyboard, not both keyboard and mouse. A disadvantage, however, is that users must remember to type them correctly, and learn the syntax of units and labels.

It would be advantageous, therefore, to have a GUI tool for entering units and labels. For units, several possibilities are conceivable. Pop-up menus, either in a separate inspector window, or in cells

when activated by editing the cells, could be used to select a unit. A sequence of these (one additional menu would be displayed when previous ones were filled in), with text boxes to display exponents, could allow users to enter products of base units.

Entering labels is somewhat harder, because the context is potentially much richer. The labels context is a hierarchy, so it would be logical to display it similarly to other hierarchies that users are accustomed to managing. For example, file systems are hierarchical, so it would make sense to use a browser paradigm in which users could choose nodes to add to a label. The system would need to report an error if the user attempted to add both a node and one of its ancestors.

6.3. Editing the Unit and Label Contexts

A separate issue from the interaction with the spreadsheet cells is the specification of the unit and label contexts.

The current implementation has a static unit context. A more complete implementation should allow users to edit unit contexts. Although a default context containing SI, English, etc. units would suffice for most users, many users in specialized settings need unique units. One important observation, however, is that in these settings, there are frequently groups of users with the same requirements. One user should be able to create a specialized unit context, and distribute it to other users by transferring a file corresponding to the unit context.

Because the unit context is very simple, editing it is not a difficult requirement. The context consists only of a list of strings; users would simply edit the list. This requirement would be complicated by the future addition of automatic conversions or unit abbreviations. Unit conversions are complex; for example, converting US dollars to Euros requires the system to know the current exchange rate.

Interaction techniques for editing the label context are more challenging than those for the unit context, since label contexts have more complicated structure. Only a small minority of users will understand trees. One possible representation would be a list of statements, as shown in Figure 5.

For some users, a hierarchical editor, similar to some file browsers, may be appropriate. One approach would have one column per level of the hierarchy; selecting an item in one column would cause the next column to display the children of the selection. The choice of trees for label contexts makes the file browser analogy fitting. Of course, such an editor would need to display errors if users ever tried to create graphs other than trees.

Just as with units, large groups of users may share a single label context, so one expert could create a context, and share it with users who do not need to understand the details of editing contexts.

Apple is a kind of **Fruit**.

Red Delicious is a kind of **Apple**.

Granny Smith is a kind of **Apple**.

…

Figure 5. Users could enter the text in bold to define a label context.

7. Future Work

Important future work would be to enrich SLATE with a more complete computational model and developing an appropriate treatment for labels in this setting, including conditional statements, Boolean operators, and a library of standard statistical functions. Likewise, one could add a more complete unit system, as have been designed in other work (such as Antoniu [3] or Kennedy [7]).

In addition to enriching the language, it is important to design unit and label context editors, and prototype a method to allow users to specify units or labels for single or multiple cells simultaneously.

Displaying units and labels requires space—sometimes more than the values they correspond to. But if labels are not visible, users must explicitly make them visible to check for errors. One approach to reducing space requirements would be to use an inspector window which would display the label of the currently selected cell. Another choice would be a heuristic that would highlight cells considered most likely to contain errors; of course, this would only be an approximation of the user's manual verification. Alternatively, labels could be displayed in tooltips when desired.

Header inference techniques from Apples and Oranges [6] could be adopted in SLATE. Instead of requiring users to enter information about objects of measurement manually, the system could infer some information from the headers, using a similar inference algorithm.

Future work should include user studies to evaluate SLATE; this is vital both for demonstrating SLATE's ability to help users detect errors and for choosing among the many design ideas discussed above. The system must also be tested with large contexts and data sets to show its effectiveness with large amounts of data.

8. Conclusions

SLATE represents a new approach to spreadsheet formula error detection. By considering objects of measurement in addition to units of measurement, it is possible to detect errors that would be hidden in other systems without presenting an onerous burden to end users. SLATE has the potential to uncover a new class of errors, and should be suitable for both small and large spreadsheets. This novel approach to error detection represents an improvement over existing techniques, which do not take objects of measurement into consideration.

9. Acknowledgments

I would like to thank Andrew Ko, Brad Myers, and Frank Pfenning for their wonderful guidance and advice throughout this project. Without their help, this project would not have come to fruition.

This research was partially supported by the EUSES consortium under NSF ITR CCR-0324770. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect those of the National Science Foundation.

10. References

[1] Abraham, Robin and Erwig, Martin. "Header and Unit Inference for Spreadsheets Through Spatial Analyses", *IEEE Symposium on Visual Languages and Human-Centric Computing*, Rome, Italy, September 2004, pp. 165-172.

[2] Y. Ahmad, T. Antoniu, and S. Goldwater. "A Type System for Statically Detecting Spreadsheet Errors", *IEEE Conference on Automated Software Engineering*, pp. 174-183, 2003.

[3] Antoniu, Tudor; Steckler, Paul A.; Krishnamurthi, Shriram, Neuwirth, Erich; and Matthias Felleisen. "Validating the Unit Correctness of Spreadsheet Programs", *Proceedings of the 26th International Conference on Software Engineering*, Scotland, UK, IEEE Computer Society, May 2004, pp. 439-448.

[4] Burnett, Margaret; Atwood, John; Djang, Rebecca Walpole; Gottfried, Herkimer; Reichwein, James; and Yang, Sherry. “Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm”, *Journal of Functional Programming*, Cambridge University Press, March 2001, pp. 155-206.

[5] Burnett, Margaret and Erwig, Martin. “Visually Customizing Inference Rules About Apples and Oranges”, *2nd IEEE International Symp. on Human Centric Computing Languages and Environments*, 2002, Arlington, VA, pp. 140-148.

[6] Erwig, Martin; and Burnett, Margaret. “Adding Apples and Oranges”, *Practical Aspects of Declarative Languages: 4th International Symposium*, Portland, Oregon, LNCS 2257, pp. 173-191, 2002.

[7] Kennedy, Andrew. “Programming Languages and Dimensions”, Ph.D. thesis published as *Technical Report No. 391, University of Cambridge Computer Laboratory*, April 1996.

[8] Panko, Raymond R. “What We Know About Spreadsheet Errors”, *Journal of End User Computing*, Idea Group Publishing, Spring 1998, pp. 15-21.

[9] Rajalingham, Kamalasen; Chadwick, David; Knight, Brian; and Edwards, Dilwyn. “Quality Control in Spreadsheets: A Software Engineering-Based Approach to Spreadsheet Development”, *Proceedings of the 33rd Hawaii International Conference on System Sciences*, Maui, Hawaii, Volume 4, 2000.

[10] Rothermel, Karen J.; Cook, Curtis R.; Burnett, Margaret M.; Schonfeld, Justin; Green, T. R. G.; Rothermel, Gregg. “WYSIWYT Testing in the Spreadsheet Paradigm: An Empirical Evaluation”, *International Conference on Software Engineering*, Limerick, Ireland, June 2000, pp. 230-239.

[11] Wilson, Aaron; Burnett, Margaret; Beckwith, Laura; Granatir, Orion; Casburn, Ledah; Cook, Curtis; Durham, Mike; and Rothermel, Gregg. "Harnessing Curiosity to Increase Correctness in End-User Programming", *ACM Conference on Human Factors in Computing Systems*, Ft. Lauderdale, April 2003, pp. 305-312.