A Model for Representing Variational Spreadsheets

Martin Erwig
Oregon State University
erwig@eecs.oregonstate.edu

Duc Le
Oregon State University
ledu@eecs.oregonstate.edu

Eric Walkingshaw
Oregon State University
walkiner@eecs.oregonstate.edu

***Abstract*—TBD**

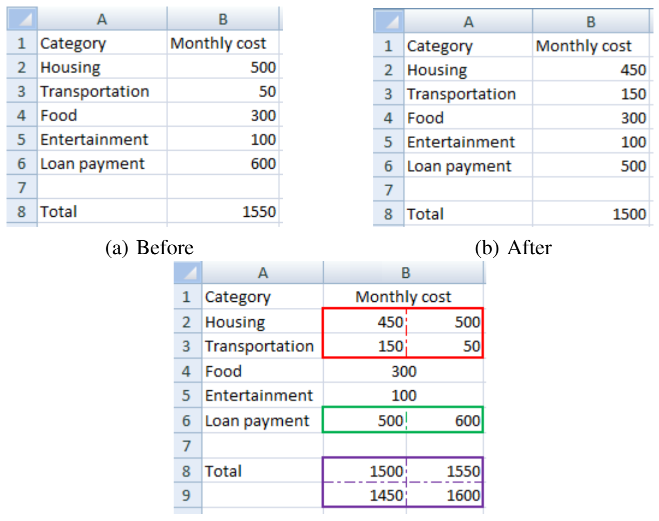
I. Introduction

New motivation needed

We first give a motivating example for VarSheet. Figure 1a shows a conventional spending estimate of a college student. Suppose the student is not happy with it, they would adjust the costs of some categories based on available options. After some adjustments, the student ends up with the spreadsheet in Figure 1b, for which they pay $50 less. In the updated spreadsheet, the housing cost is $50 less since the rented house is further from campus, but that would increase the cost of transportation. The student also decides to pay $100 less on their loan. Considering both versions, the student would probably go after the latter one, and by doing this, they lose the chance to reduce another $50. Had the student chosen the original housing option and kept Loan payment to be $500, the monthly cost would have been $1450.

Figure 1c shows a possible user interface of VarSheet that could support the student in their decision making process. There are three variation points in the spreadsheet. The *Housing&Transportation* categories are related and thus grouped into a red box with a dashed line separating the two available options. In VarSheet, *Housing&Transportation* is called a *dimension*—a choice users have to make. A dimension contains a set of options, each being called a *tag*. *Housing&Transportation*'s tags are *Close* and *Far*. The *Loan Payment* dimension represents a different variation point with two tags *500* and *600* and thus is colored differently (green). The last variation point, the *Total* category, does not contain variational formulas by itself. The formula being used is `SUM(B2:B6)` and is non-variational, yet the cell inherits the variational structures of its referred cells and hence contains four available options. This cell is therefore colored purple to indicate that it contains *induced variation*. Showing all the four alternatives of *Total* gives the student an overview of all the different options they have. Moreover, if the student selects the $1450 alternative, VarSheet will automatically make decisions for *Housing&Transportation* and *Loan Payment*, and displays those decisions and the resulting spreadsheet to them. We call this feature *goal-directed selection*, the process of selecting spreadsheet variants based on certain goals.

The study of variational spreadsheets brings up several insights to current research on software variation. While traditional variation mechanisms focus on either the syntax or semantics domain, spreadsheets' immediate semantics computation expands the application of variational constructs to both domains, enabling the realization of goal-directed selection. In the example above, Loan payment varies syntactically while Total varies semanticallly, and one could even define cells that vary on both domains.



	A	B
1	Category	Monthly cost
2	Housing	500
3	Transportation	50
4	Food	300
5	Entertainment	100
6	Loan payment	600
7		
8	Total	1550

(a) Before

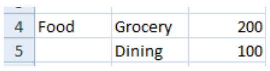
	A	B
1	Category	Monthly cost
2	Housing	450
3	Transportation	150
4	Food	300
5	Entertainment	100
6	Loan payment	500
7		
8	Total	1500

(b) After

	A	B	
1	Category	Monthly cost	
2	Housing	450	500
3	Transportation	150	50
4	Food	300	
5	Entertainment	100	
6	Loan payment	500	600
7			
8	Total	1500	1550
9		1450	1600

(c) A Possible User Interface

Figure 1: A Monthly Spending Spreadsheet



4	Food	Grocery	200
5		Dining	100

Figure 2: A Different Way to Represent the Food Category

Existing variational contructs mainly work on linear or tree structures, which localizes their scope of impact. Spreadsheets have a special two dimensional structure that makes localization hard to achieve. For instance, in Figure 1a, one could replace row 4 by the spreadsheet in Figure 2 and expects this variation introduction to be local. This action unfortunately has a global impact on the spreadsheet's structure and the addresses/values of several unrelated cells. The Monthly Cost column has to be shifted to column C, while the Total column has to be shifted one row down. In our variational spreadsheet model, we provide mechanisms to localize the effect of structural changes.

Lastly, by letting users actively define variation in spreadsheets, we remove the need for using spreadsheet diffing algorithm, which can be imperfect and misleading at times.

II. Background and Related Work

To be rewritten

Talk about the prototype of the VLHCC 2011 paper

Existing empirical research demonstrates the need for an effective approach to deal with spreadsheet variation. Spreadsheet reusing is common, but users often have to choose from various options [citation needed]. Once a spreadsheet is chosen and several modifications have been made to it, if users recognize that it was not the right one to begin with, they will have to start all over again on a different one. This could happen for many times until users are happy with their choice. To mitigate this problem, VarSheet gives users the ability to modify multiple versions/spreadsheets at the same time on a single representation and select a desired version later. Another reason for spreadsheet variation is due to spreadsheet errors and debugging. Spreadsheets contain errors [8] [9], many of which are introduced in the process of reusing and modifying existing spreadsheets. When debugging, users often need to show the differences between multiple versions, so a framework for systematically managing changes is needed.

In the area of spreadsheet change support, existing tools can be classified into two big categories: change tracking tools and spreadsheet diffing tools. One representative example of a change tracking tool is Microsoft Excel's change tracking feature, which provides users the ability to track spreadsheet edits such as inserting rows, updating equations, etc. This tool is useful and effective in helping users understand versioning information of spreadsheets but is not without limitation. The entire variational spreadsheet is represented using only the time dimension. It is not possible to group changes into categories or groups such that they can be undone or applied again. Another problem arises when two or more users copy and modify the same original spreadsheet. When trying to merge the modified copies, it is unclear which change includes or excludes other changes. For VarSheet, grouping changes could simply be resolved by changing dimension/choice names, which will be defined in the next sections. Research and commercial tools for diffing spreadsheets are prevalent, including CC DiffEngineX [1] and Synkronizer [2]. These tools are effective in comparing spreadsheets and producing accurate results. However, they do not reveal the original purposes of users' changes and do not provide a way to document those.

On a broader topic, there has been extensive research on the topic of representing and managing software variation. The two big pillars of this topic is the compositional approach [6], which modularizes software product line features [4] into individual folders and describes variability at a higher level using feature models [3], and the annotated approach [6], where variability is encoded and represented inside source code. Since there are advantages and disadvantages for each approach, Erwig and Walkingshaw [5] designed the Choice Calculus to shorten the gap between them and to take advantage of the approaches' benefits. The Choice Calculus's design is based on the idea that software variation should be done at both source code and higher levels with not-too-restrictive and not-too-relaxed constraints, making it highly applicable for tree-like structures. VarSheet expands Choice Calculus's ideas of dimensions and choices to the spatial, two dimensional structure of spreadsheets.

III. VarSheet's Design

VarSheet's design follows the annotative approach and encodes variation at the most fine-grained level—the cell level. Every single cell is annotated with a description about which variant it belongs to. A variational spreadsheet represents a universe of annotated cells from which one could pick subset to form a variant. By convention, users only see individual variants, yet they can go into *variational exploration mode* to explore all alternatives of a cell or a set of cells (as in Figure 1c). Each cell is associated with a distinct global identity. Formally, a variational spreadsheet is a partial function from the set of global identities to sets of tags, *relative spatial positions*, and formulas.

$$s \in S = id \rightarrow (2^t, p, f) \tag{1}$$

In the above definition, s stands for variational spreadsheets, id ranges over global identities[1], t stands for tags, 2^t is the power set of all tags, p represents relative partial positions (more on this later), and f represents formulas. Each cell can then be encoded as a tuple (id, 2^t, p, f).

A. Dimensions and Tags

Definition A *dimension* consists of a set of options, each being called a *tag*.

We distinguish tags having identical names by qualifying them with dimension names (e.g. *Housing&Transportation.Close*). Dimensions and tags form the basis for how variational spreadsheets vary. Every variational spreadsheet can constitute several independent dimensions, each of which represents an independent choice users have to make. Every cell contains tags to decide which variant the cell belongs to.

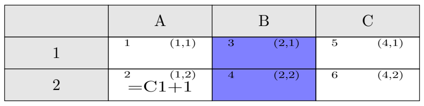
Definition A *decision* is a set of tags from different dimensions. A *complete decision* is a decision with tags from all dimensions in a variational spreadsheet.

A decision can be an empty set (∅), a set of a single tag ({*Housing&Transportation.Close*}), or a set of multiple tags ({*Housing&Transportation.Close, Loan Payment.500*}). Decisions provide means for selecting individual variants from variational spreadsheets. Given the decision {*Housing&Transportation.Close, Loan Payment.600*}, the variant in Figure 1a is selected. Given the decision {*Housing&Transportation.Far, Loan Payment.500*}, the variant in Figure 1b is selected.

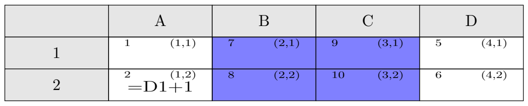
In Figure 3 we provide two variants of a variational spreadsheet containing a single dimension *D* with two tags *D.1* and *D.2*. As the consequence, there are two decisions, {*D.1*} and {*D.2*}, the former associating with variant 1 while the latter with variant 2. On the top-left corner of each cell is its global identity. We leave the contents of several cells blank as they are not important for our discussion. Two different types of cells exist in Figure 3, *non-variational* and *variational* cells. Non-variational cells are cells that appear in all variants whereas variational cells do not. Cells with ID 1, 2, 5, 6 are non-variational (from now on we will use the convention: cell #1, #2, #5, #6), and all other cells are variational ones. Non-variational cells' tags are empty sets while variational cells' tags are non-empty. We provide the cells' tags below.

#1,#2,#5,#6	: ∅
#3,#4	: {*D.1*}
#7,#8,#9,#10	: {*D.2*}

[1]We assume the set of global identities is an infinite set from which fresh, unused identities can be drawn. In this paper, we use natural numbers to represent global identities.



(a) Variant 1



(b) Variant 2

Figure 3: Two variants of a variational spreadsheet

B. Relative Spatial Positions and Formulas

Figure 3 gives an example of *structural variation*, where a 2x1 sub-spreadsheet in variant 1 was replaced by a 2x2 sub-spreadsheet in variant 2. In each variant, there is a reference from the cell #2 to the cell #5. The formula of cell #2 changes when we switch from one variant to the other (from `=C1+1` to `=D1+1` and vice versa), which surprisingly implies that cell #2 is influenced by the decision in dimension D, which should not be the case. This results from the fact that structural changes in spreadsheets have global impact, and we need a mechanism to reduce the scope of impact to the minimal.

Instead of using conventional spreadsheets' absolute/relative address referencing schemes, we employ in our model a *global identity referencing scheme* in which the formula of #2 is `=#5+1`. When the first variant of the spreadsheet is shown, #5's address is mapped to `C1`, and when the second variant is selected, #5's address is mapped to `D1`, so the formula of #2 is displayed according to the selected variant. The definition for spreadsheet formulas is thus defined as

$$\begin{array}{rcll} f \in F & ::= & v & \textit{values} \\ & | & id & \textit{identity references} \\ & | & \omega(f,\ldots,f) & \textit{functions} \end{array}$$

Note that we store identity references but show normal address references to users. As cell definitions do not encode address information, the burden now lies on how to map cell identities to addresses. We solve this problem by defining a pretty printing algorithm for variants. The process of selecting a variant is similar to picking a subset of cells from the universe of cells. Each cell in the subset carries information about its *relative spatial position*, which is used to aid the pretty printing algorithm in deciding cell addresses. Relative spatial positions are pairs of natural numbers, p = ($\mathbb{N}$, $\mathbb{N}$), the first element representing relative vertial positions and the second representing relative horizontal positions. In Figure 3, spatial positions are shown on the top-right corner of each cell. Normal cell addresses are pairs of strings and natural numbers, $a = (\mathbb{S}, \mathbb{N})$.

The pretty printing algorithm (presented in Algorithm 1) takes a set of cells as input and attaches an addresses to each cell (e.g. #1 will have the address `A1`). This algorithm assumes the left-most and top-most cell's address is `A1`.

IV. Semantics

The pretty printing algorithm works on individual variants, which are selected by picking a subset of cells from the cell universe. A natural question is how do we know which cell to pick. To answer this question, in this section we describe *variation semantics*, a mapping between complete decisions

```
input : A set C of cells, each being encoded with ($id$, $2^t$, $p$, $f$)
output: A mapping between cells and addresses
1  Extract the set of positions P from C;
2  The set of vertical positions VP = $\{x \mid (x,y) \leftarrow P\}$ ;
3  The set of horizontal positions HP = $\{y \mid (x,y) \leftarrow P\}$ ;
4  Sort elements in VP and HP in ascending order ;
5  Tag the first element in VP with "A", the second with "B" and so on;
6  Tag the first element in HP with 1, the second with 2 and so on;
7  ret ← ∅;
8  for c ∈ C do
9      v ← c's vertical position;
10     h ← c's horizontal position;
11     c's column address cadd ← tagLookUp(VP, v);
12     c's row address radd ← tagLookUp(HP, h);
13     ret ← ret ∪ {(c, (cadd, radd))};
14 end
15 return ret;
```

Algorithm 1: Pretty Printing Spreadsheet Variants

and spreadsheet variants.[2]

The steps involved in computing variation semantics are: (1) collecting the set of dimensions, (2) generating all complete decisions, and (3) performing *tag selection* on those decisions to produce variants.

A. Dimensions and Complete Decisions

We define the *dims* operation for collecting all dimensions in a variational spreadsheet. For the spreadsheet in Figure 1, *dims* returns {*Housing&Transportation*, *Loan Payment*}. The set of complete decisions of a variational spreadsheet s is defined as

$$decisions(s) = \{D_1.t_1, \ldots, D_k.t_k \mid \{D_1, \ldots, D_k\} = dims(s), t_i \leftarrow D_i\}$$

B. Tag Selection

The tag selection procedure $\lfloor s \rfloor_{ts}$ takes a complete decision ts and goes through a variational spreadsheet s to filter out all cells whose tags are subsets of ts into a variant.

$$\lfloor s \rfloor_{ts} = \{c \mid c \leftarrow s, tags(c) \subseteq ts\}$$

Notice the *tags* operation, which selects a variational cell's tags .

Variation semantics is then defined as

$$V(s) = \{(ts, \lfloor s \rfloor_{ts}) \mid ts \leftarrow decisions(s)\}$$

[2]Note that we ignore the discussion about the semantics of individual variants since they are basically the semantics of plain spreadsheets.

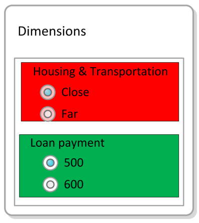


Figure 4: Dimension Panel

For the spreadsheet in Figure 3, the decisions are $\{D.1\}$ and $\{D.2\}$. Given each cell's tags, $\{D.1\}$ is mapped to the set of cells {#1, #2, #3, #4, #5, #6}, and $\{D.2\}$ is mapped to the set of cells {#1, #2, #5, #6, #7, #8, #9, #10}. These two sets then become inputs for the pretty printer to generate cell addresses.

V. Concrete Syntax

The concrete syntax has two different modes: the *variation exploration* mode and the default *individual variant* mode. In the variation exploration mode, users can expand several variation points to show all the available alternatives as in Figure 1c. Users can also select a certain alternative of a variation point and corresponding decisions will be made automatically. The individual variant mode displays individual variants to users. To see a variant, users make decisions using the dimension panel as in Figure 4, which visualizes the relationship between dimensions and tags. All tags of a dimension are grouped into a radiobutton group. Users make decision for that dimension by selecting one of the radiobuttons. The idea of dimension panel is reused from our previous work on variational representation of source code [7]. This work confirmed that showing individual variants and separating the configuration structures into a dimension panel improved program comprehension.

Each dimension is mapped to a distinct color and all cells relating to that dimension is annotated with the same color. This matching is only applicable when a cell's tags has only one dimension. When a cell is associated with two or more dimensions, we could apply different visual cues to represent the relationship. One approach could be creating a color palette at one of the cell's corner for storing the colors of all associated dimensions. Another approach is using a special color for the cell to indicate its dependence on multiple dimensions and then when the cell is highlighted, all related dimensions are also highlighted in the dimension panel.

Producing the concrete presentation of spreadsheet variants requires taking the output of the pretty printer and positioning cells at appropriate addresses. We require that each variant has a rectangular shape, which results in cases where one has to add *filler cells* to fill up empty spaces. Table I gives an example where filler cells are needed. There are four variants of a variational spreadsheet with two dimensions $D = \{D_1, D_2\}$ and $D' = \{D'_1, D'_2\}$. The column and row headers represent the corresponding decision of each variant. For instance, the decision $\{D_1, D'_2\}$ corresponds to the lower left variant. Each cell's tags are listed below.

#1	:	$\{D_1\}$
#2	:	$\{D'_1\}$
#3, #4	:	$\{D_1\}$
#5, #6	:	$\{D'_2\}$

Cell #7 and #8 are filler cells, that is, they are not represented or stored in our model, yet they are needed to ensure variants' rectangular shapes. Cell #7 is added whenever D'_2 is chosen. For example, our semantics function maps the decision $\{D_1, D'_2\}$ to the set of three cells {#1, #5, #6}. The pretty printer then generates the corresponding addresses for each of these cells.

#1 :	`A1`
#5 :	`B1`
#6 :	`B2`

Since address `A2` is not occupied by any cell, it becomes a "black hole" and is filled with cell #7. While cell #7 is added when a singular choice D'_2 is chosen, #8 requires two choices D_2 and D'_2 to be added to corresponding variants.

TODO: Get rid of relative postions from #7 and #8 in Table I

VI. Partial Tag Selection and Goal-Directed Selection

Variation semantics works with complete decisions, which is applicable when users could make decisions for all dimensions. In practice, users might not know all decisions in advance. They might make decisions for some dimensions, observe the outcome, and then either undo or reiterate the process. We introduce the notion of *partial tag selection* to support users in making partial decision. Partial tag selection takes a variational spreadsheet and a decision as input and returns another variational spreadsheet. When the input is a complete decision, partial tag selection returns a variant. For instance, when applying the incomplete decision $\{Housing\&Transportation.Close\}$ on the spreadsheet in Figure 1, we obtain a spreadsheet with *Loan Payment* as the only dimension and the costs of housing and transportation become non-variational.

Formally, the partially tag selection operation $\lfloor\rfloor^p : S \times 2^t \to S$ is defined as

$$\lfloor s \rfloor^p_{ts} = \{update(c, ts) \mid c \leftarrow s, tags(c) \sim ts\}$$

The *update* operation removes all tags in ts from cell c's tags. If $tags(c) \subseteq ts$, c's tags becomes empty and the cell becomes non-variational. The $\sim$ relation returns true if two sets of tags do not have conflicts in decisions, that is, for every shared dimension, corresponding tags must be the same. For instance, $\{D_1.t_1, D_2.t_2\} \sim \{D_1.t_1\}$ holds while $\{D_1.t_1, D_2.t_2\} \sim \{D_1.t'_1, D_2.t_2\}$ does not.

$$ts_1 \sim ts_2 = \forall D, D.t_1 \in ts_1 \wedge D.t_2 \in ts_2 \to t_1 = t_2$$

Goal-directed selection is defined as partial tag selection on variational spreadsheets. Users begin by inspecting all possible values of a cell. In Figure 1c, the cell being inspected is `B8` and has four possible values, each corresponding to a partial decision (In this example specifically, the partial decisions happen to be the complete decisions). The alternative 1450 corresponds to the decision $\{Housing\&Transportation.Close, Loan\ Payment.500\}$, which could be applied to the spreadsheet to obtain a single variant.

We simplify our discussion by looking four variants of a simpler spreadsheet in Table II...

	D_1			D_2			
		A	B		A	B	C
D'_1	1	1 (1,1)	2 (3,1)	1	3 (1,1)	4 (2,1)	2 (3,1)
D'_2	1	1 (1,1)	5 (3,1)	1	1 (1,1)	4 (2,1)	5 (3,1)
	2	7 (1,2)	6 (3,2)	2	7 (1,2)	8 (2,2)	6 (3,2)

Table I: Four Variants of a Spreadsheet

	D_1			D_2		
		A	B		A	B
D'_1	1	1 Price (1,1)	4 100 (2,1)	1	1 Price (1,1)	5 110 (2,1)
	2	2 Tax (1,2)	6 0.06 (2,2)	2	2 Tax (1,2)	6 0.06 (2,2)
	3	3 Total Cost (1,3)	8 =#4*(1 + #6) (2,3)	3	3 Total Cost (1,3)	9 =#5*(1 + #6) (2,3)
D'_2	1	1 Price (1,1)	4 100 (2,1)	1	1 Price (1,1)	5 110 (2,1)
	2	2 Tax (1,2)	7 0.09 (2,2)	2	2 Tax (1,2)	7 0.09 (2,2)
	3	3 Total Cost (1,3)	10 =#4*(1 + #7) (2,3)	3	3 Total Cost (1,3)	11 =#5*(1 + #7) (2,3)

Table II: Four Variants of the Receipt Spreadsheet

TODO: (0) Provide a simpler example to talk about goal directed selection and syntactic sugar, (1) Talk about syntactic sugar (address referencing) here to enable goal-directed selection, (2) Describe goal-directed selection in more details, (3) Syntactic revisit: how to show all variants of a cell at a certain address?

VII. Language Properties

Theorem 1. *If $ts_1 \sim ts_2$ and $ts_1 \cap ts_2 = \varnothing$,*

$$\lfloor s \rfloor_{ts_1 \cup ts_2} = \lfloor \lfloor s \rfloor^p_{ts_1} \rfloor^p_{ts_2} = \lfloor \lfloor s \rfloor^p_{ts_2} \rfloor^p_{ts_1}$$

Proof: ... ■

TODO: Add three more theorems: (1) C-S equivalent (choice expansion), (2) Reduction of tags, (3) Dimension dependency theorem

VIII. Conclusion and Future Work

TBD

References

[1] Florencesoft diffenginex compares microsoft excel worksheets. http://www.florencesoft.com/exceldiff.html, 2012.

[2] Synkronizer compares excel files faster than you can. http://www.synkronizer.com/, 2012.

[3] D. Batory. Feature Models, Grammars, and Propositional Formulas. In *Int. Software Product Line Conf.*, volume 3714 of *LNCS*, pages 7–20. Springer-Verlang, 2005.

[4] P. C. Clements and L. M. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, 2001.

[5] M. Erwig and E. Walkingshaw. The Choice Calculus: A Representation for Software Variation. *ACM Trans. on Software Engineering and Methodology*, 2011.

[6] C. Kästner and S. Apel. Integrating Compositional and Annotative Approaches for Product Line Engineering. In *GPCE Workshop on Modularization, Composition and Generative Techniques for Product Line Engineering*, pages 35–40, 2008.

[7] D. Le, E. Walkingshaw, and M. Erwig. #ifdef confirmed harmful: Promoting understandable software variation. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 143 –150, 2011.

[8] R. R. Panko. What we know about spreadsheet errors. *Journal of End User Computing*, 10:15–21, 1998.

[9] S. G. Powell, K. R. Baker, and B. Lawson. A critical review of the literature on spreadsheet errors. *Decision Support Systems*, 46(1):128 – 138, 2008.