Template Parsing With User Feedback*

Robin Abraham
Oregon State University
abraharo@eecs.oregonstate.edu

1. Introduction

Spreadsheets are among the most widely used end-user programming systems. According to some estimates, up to 90% of spreadsheets have non-trivial errors in them [7]. In many cases, spreadsheet errors have resulted in huge financial losses for companies. Spreadsheets are also in use in Science and Mathematics education in schools primarily because they offer a flexible modeling environment. With widespread adoption of spreadsheets by end users there is a greater need for better auditing tools to help users develop safer spreadsheets. Traditional approaches like code inspection, auditing, and testing might not be really conducive for users disadvantaged by their background, education, learning style or physical abilities. In this context, it is important to develop tools that work with minimal user intervention.

Errors in spreadsheets might arise for a variety of reasons ranging from the user's lack of understanding of the specifications or requirements of the spreadsheet to errors arising from entering the formulas or values incorrectly (for example, typos, poor understanding of operator precedence etc.). In Section 2 we discuss the main approaches we have explored so far. In Section 3 we discuss the importance of being able to infer the underlying models of spreadsheets and the steps involved in a first attempt at extracting specifications (templates) from spreadsheets.

2. Prevention, Detection, and Correction of Spreadsheet Errors

Gencel: In the approach described in [5], the users start with a spreadsheet specification in the form of a ViTSL template [3] and work towards building their spreadsheets using the update operations provided by the Gencel system. These update operations, customized according to the template being used by the end user, guarantee that all evolutions of the spreadsheet will match the initial specification and be free from formula and type errors.

UCheck: We have also looked at the automatic detection of so called “unit errors”. In the first step, the UCheck system [1] infers the headers entered by the users to label their data. The system then uses the inferred headers to assign units to all the cells in the spreadsheet. The assigned units are then used to identify unit-inconsistent computations (formulas that violate the allowed unit combinations) within the spreadsheet and these are then reported as unit errors to the users.

Spreadsheet Debugger: We have developed a debugger that allows the user to mark a cell with an error and specify the expected value. The system then generates a list of possible changes that would result in the marked cell evaluating to the expected value [2]. The user can select and apply a particular change suggestion, reject suggested changes, or ask the system to generate more suggestions.

3. Template Inference in Spreadsheets

All three of the systems described in the previous section could benefit from information about the underlying specification, or model, of the spreadsheet.

In the current version of Gencel, users have to start with a ViTSL specification and create their spreadsheets. This approach, as it is currently implemented, has two main shortcomings. Firstly, there is no easy way for users to port their existing Excel spreadsheets to Gencel. Secondly, users can not switch back and forth between editing their specifications and spreadsheets. Both of these problems could be solved if the system had the facility to infer the specifications automatically.

The rules for unit inference in UCheck are conservative to guard against false negatives. The header inference process would become more accurate with knowledge of the underlying template. Information about the template would also allow us to relax the unit inference rules in some cases to minimize false positives.

The current version of the spreadsheet debugger uses inverse functions to generate suggested changes to formulas to obtain the target value specified by the user. The system then uses a set of heuristics to rank the generated change suggestions and presents them to the user, ordered by their rank. Knowledge of the underlying template would enable the system to rank the suggestions better. Moreover, cells or regions that do not fully agree with the inferred template could be considered potential hotspots for errors.

An example spreadsheet and its corresponding template are shown in Figure 1 and Figure 2 respectively. In the template, the absence of borders between the column names B, C, and D is meant to indicate that they form a group that can be repeated horizontally (indicated by the horizontal dots between D and E). We refer to such horizontally-repeatable groups of columns as hex groups. The hex group in this case stores the quantity, cost per unit, and total cost information for each year. Similarly, the vertically aligned dots between row 3 and 4 are meant to indicate that row 3 can be vertically repeated downwards. We refer to the vertically repeatable groups of rows as vex groups. The vex group in this case stores information about elements in the category. This ViTSL template can be loaded into Gencel and the spreadsheet shown in Figure 1 can be produced by inserting one

* This work is supported by the National Science Foundation under the grant ITR-0325273 and by the EUSES Consortium [4].
column for an additional year, two rows for the two items, and entering in the corresponding values.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Category</td>
<td>Only</td>
<td>Cost</td>
<td>Total</td>
<td>Cost</td>
</tr>
<tr>
<td>2</td>
<td>Students</td>
<td>2</td>
<td>22000</td>
<td>=B3*C3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Computers 2</td>
<td>1500</td>
<td>=B4*C4</td>
<td>3</td>
<td>625</td>
</tr>
<tr>
<td>4</td>
<td>Travel</td>
<td>3</td>
<td>1000</td>
<td>=B5*C5</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>Total</td>
<td>3</td>
<td>1000</td>
<td>=SUM(D3:D5)</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 2: ViTSL specification for the budget sheet

Extraction of the spreadsheet template from a given spreadsheet can be a highly ambiguous task. This problem arises because spreadsheets do not impose any restrictions on how the users map their mental models to the two-dimensional grid framework (flexibility is one of the main reasons for the popularity of spreadsheets.) Because of this ambiguity problem, we suspect that template inference cannot be fully automated. As a first attempt, we are looking at a three-step approach to template parsing. Each step would accept user-directed refinements to guide it.

Identifying Tables in Spreadsheets The header inference process in the UCheck system identifies hard fences (empty cells that separate the different regions of the spreadsheet) and uses this information to break up the spreadsheet into tables. Leveraging header inference from UCheck would also help us to identify the different regions (header cells, core/input cells, and footer cells) within tables. Formatting information might provide additional clues about tables.

Identifying Regions We plan to adopt strategies similar to the ones described in [6] to identify regions within tables. We employ this information about regions (and maybe even nested regions) as clues to the boundaries of repeating groups.

Inferring Specifications After regions have been recognized, contained repeating groups can be identified (in a subsequent step). For example, rows that are “copies” of each other (converting to only relative references in formulas) could be considered as instances of the same vex group.

User input during template parsing is very important because, in general, there might be more than one “correct” parse for a given spreadsheet. We will use heuristics to rank the generated parses. For example, the size of the generated specification could be used as a measure of success. A specification with the least number of cells would get the highest rank in this case. Another problem is that, in some cases, the spreadsheet being considered might not have enough information for the correct specification to be inferred. For example, in the spreadsheet shown in Figure 1, if the information for the year 2006 (i.e., columns E, F, and G) were not present, the parser might be able to successfully identify the vex group but it simply does not have information to identify the hex group shown in Figure 2. Incorrect formulas within the spreadsheet might also lead to the system inferring incorrect templates.

Template parsing would require some extra effort from the end user in addition to the creation of the spreadsheet, but it offers automatic conformance to the template, automatic consistency checking, and help with spreadsheet debugging in return. Knowledge of the template would also help the system generate better error messages and context-sensitive examples. We believe that these features would allow end users to focus more on their actual tasks and less on cell-level spreadsheet interactions in the long run.

References


