Detecting Faulty Empty Cells in Spreadsheets

Liang Xu[1,2], Shuo Wang[2,3],Wensheng Dou[1,2*], Bo Yang[3], Chushu Gao[1,2], Jun Wei[1,2], Tao Huang[1,2]
[1]University of Chinese Academy of Sciences, China
[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
[3]North China University of Technology
[2]{xuliang12, wangshuo, wsdou, gaochushu, wj, tao}@otcaix.iscas.ac.cn, [3]yangbo090313@163.com

***Abstract*—Spreadsheets play an important role in various business tasks, such as financial reports and data analysis. In spreadsheets, empty cells are widely used for different purposes, e.g., separating different tables, or default value "0". However, a user may delete a formula unintentionally, and leave a cell empty. Such ad-hoc modification may introduce a faulty empty cell that should have a formula.**

We observe that the context of an empty cell can help determine whether the empty cell is faulty. For example, is the empty cell next to a cell array in which all cells share the same semantics? Does the empty cell have headers similar to other non-empty cells'? In this paper, we propose *EmptyCheck*, to detect faulty empty cells in spreadsheets. By analyzing the context of an empty cell, EmptyCheck validates whether the cell belong to a cell array. If yes, the empty cell is faulty since it does not contain a formula. We evaluate EmptyCheck on 100 randomly sampled EUSES spreadsheets. The experimental result shows that EmptyCheck can detect faulty empty cells with high precision (75.00%) and recall (87.04%). Existing techniques can detect only 4.26% of the true faulty empty cells that EmptyCheck detects.

Keywords-Spreadsheet, fault, empty cell

I. INTRODUCTION

Spreadsheets are widely used in financial reports, data storage and analysis. A recent study shows that, in Europe, 80% of the companies use spreadsheets to write their financial reports, and in America, 95% of the companies use spreadsheets in their business [1]. In 2002, there were more than 55 million users who worked with spreadsheets in the United States alone [2].

Faults / errors can be easily introduced into spreadsheets due to ad-hoc modifications [3]. Various techniques have been developed to improve the quality of spreadsheets, such as testing [4][5], fault detection [6][7][8][9][10], clone detection [11][12] and debugging [13][14].

In spreadsheets, empty cells are usually used for different purposes. Fig. 1 shows an example about empty cells. (1) Empty cells can be used to separate different parts of data and formulas, such as empty cells in rows 13 and 18. (2) Empty cells can be used to form table layouts, such as cells [A2:A4]. (3) Empty cells can be used as default value "0", such as cells B5 and B6. All the above three cases are common in practice. Note that, empty cells in the first two cases have nothing to do with business logic in the spreadsheets.

* Corresponding author

However, not all empty cells should be left as they are. For some empty cells, their contexts show that they should contain some formulas. For example, in Fig. 1, column G calculates the total parcel tax by adding the data in columns B, C, and E. However, cell G8 is left empty wrongly. Two possible scenarios can cause this: (1) The user deleted the formula in cell G8 unintentionally, because its input cells (i.e., B8, C8 and E8) were not filled with any values. (2) The user inserted a new row (i.e., row 8) and forgot filling the formula in cell G8. Similarly, cells G11 and G12 should also contain formulas. Since these empty cells (e.g., G8, G11 and G12) do not contain formulas, any update to their input cells can cause wrong values. For example, the value of cell G11 should be 0.5, because one of its input cells, i.e., E8, contains the value 0.5. However, cell G11 only contains the default value "0". Note that, cell G13 should not contain any formula, since row 13 is used to separate the whole table.

In this paper, we propose *EmptyCheck*, to detect faulty empty cells in spreadsheets automatically. The key challenge is to distinguish which empty cells are faulty and which are not. We observe that the context of an empty cell may help determine whether an empty cell is faulty. For example, if an empty cell is next to a cell array, in which all cells share the same semantics, the empty cell is likely to be faulty. For example, in Fig. 1, empty cell G8 is next to the cell array [G5:G7] and likely to be part of this cell array with a formula. Based on this observation, we propose a novel cluster-based algorithm to detect faulty empty cells in spreadsheets. We first try to identify all cell arrays by several heuristic rules e.g., all formulas contained in a cell array should reference their input cells in the same way. For each empty cell, we check whether it can be clustered into a cell array according to its context. If we find a cell array for an empty cell, there is a high possibility that the empty cell should contain a formula, thus we mark it faulty.

We implement EmptyCheck as an Excel plugin and evaluate it on 100 spreadsheets, which were randomly selected from the EUSES corpus [15]. Our experimental results show that: (1) Faulty empty cells are common in the real-life spreadsheets, and 36% of the spreadsheets we studied contain faulty empty cells. (2) 72.02% of the faulty empty cells contain wrong values. This indicates faulty empty cells are indeed harmful. (3) EmptyCheck can detect faulty empty cells with high precision (75.00%) and recall (87.04%). (4) EmptyCheck outperforms the existing techniques [16] in detecting faulty empty cells. Only 5.39% of the faulty empty

	A	B	C	D	E	F	G
1	GRADES 9-12	1999 PARCEL TAX PROGRAM			1987 PARCEL TAX PROGRAM		TOTAL OF BOTH
2		FTE		COST	FTE	COST	PARCEL TAX
3		New	Continuing	2001-02	Continuing	2001-02	PROGRAMS
4	Science		0.6	48653	0.8	66991	=B5+C5+E5
5	Math		0.4	28812	0.4	32982	=B6+C6+E6
6	Music/Performi	0.6	1.6	145056	1	73552	=B7+C7+E7
7	Performing			3000		5000	
8	Technical/Voca				1.4	108784	=B9+C9+E9
9	Athletics		0.4	33175			=B10+C10+E10
10	AHS Athletic				0.5	116761	
11	AHS/AMS			10000			
12							
13	Library	0.2	0.8	70084			=B14+C14
14	(Classified)		0.4	9986			=B15+C15
15	Counseling		2.6	221035			=B16+C16+E16
16	Foreign Langua		0.4	26466			=B17+C17+E17
17							
18	Writing				0.6	42030	=B19+C19+E19
19	TOTALS:	=SUM(B5:B19)	=SUM(C5:C19)	=SUM(D5:D19)	=SUM(E5:E19)	=SUM(F5:F19)	=SUM(G5:G19)

Fig. 1. A real world spreadsheet snippet extracted from the EUSES corpus [15]. The empty cells marked by a red right-cornered triangle are faulty.

cells can be detected by the existing techniques. We have made all the data used in our study available online for further study (http://www.tcse.cn/~wsdou/project/EmptyCheck/).

Although various techniques have been proposed to detect faults and smells in spreadsheets, e.g., AmCheck [6] / CACheck [7], TableCheck [12] and CUSTODES [8], they usually consider empty cells as boundaries for other cells, and do not check their validity. EmptyCheck differs from the faulty empty cell detection proposed by SmellSheet Detective [16] in the definitions of faulty empty cells and proposed approaches. SmellSheet Detective considers an empty cell faulty when its four neighboring cells in the same row or column are all non-empty. For example, SmellSheet Detective marks cell E8 in Fig. 1 faulty, because cell E8's four neighboring cells (i.e., E5, E6, E7 and E9) are not empty. SmellSheet Detective cannot detect the faulty empty cell G12, because its neighboring cells, i.e., G11 and G13, are empty. We can see that SmellSheet Detective can easily miss true faulty empty cells (e.g., cells G11 and G12) and return false positives (e.g., cell E8). Our experimental result shows that the neighbor-based approach [16] detected faulty empty cells with low precision (5.39%) and recall (3.70%).

In general, the main contributions of this paper are as follows:

- We propose a cluster-based approach, EmptyCheck, to detect faulty empty cells in spreadsheets automatically, by analyzing the contexts of empty cells.
- We implement EmptyCheck, and evaluate it on the real world spreadsheets from the EUSES corpus. The experimental result shows that EmptyCheck can detect faulty empty cells with higher precision and recall.
- We compare EmptyCheck with existing techniques. Our experimental result indicates EmptyCheck can accurately detect more faulty empty cells than existing techniques.

The remainder of this paper is organized as follows: Section II shows our motivation and challenges. Section III gives the detailed description of EmptyCheck. Section V shows the results of our evaluation. We discuss the limitations in Section VI and threats in Section VII and related work in Section VIII. Finally, we conclude this paper in Section IX.

II. MOTIVATION

In this section, we present a real world spreadsheet example extracted from the EUSES corpus [15]. Through this example, we first illustrate what faulty empty cells are. We further show the challenges in detecting faulty empty cells.

A. Motivating Example

Fig. 1 shows a real world spreadsheet snippet extracted from the EUSES corpus [15]. This spreadsheet snippet is used to calculate the parcel tax. It contains many empty cells. We cluster these empty cells into two categories according to the degree of their harmfulness as follows.

Harmless empty cells. Users may leave some cells empty intentionally. First, users may insert empty rows or columns to make the data layout more intuitive, such as rows 13 and 18 in Fig. 1. Second, users may leave the cells as empty when the corresponding data is not available (e.g., cells B5 and B6). Their values need to be entered directly by users in the future, or just leave as default value "0". In this paper, these empty cells are considered as correct and harmless ones.

Faulty empty cells. Some empty cells should be filled with formulas to make sure that their values can be updated automatically when their input cells' values change. For example, as shown in Fig. 1, we can recognize that the cells in column G are used to calculate the parcel tax by adding columns B, C and E. Although the value of G8 is correct since its input cells B8, C8 and E8 are empty, cell G8 should be filled with a formula "=B8+C8+E8", so that it can be updated automatically when the new data are filled into cells B8, C8 or E8. We consider cell G8 as a faulty empty cell since it should contain a formula. Similarly, cells G11 and G12 are also faulty empty cells.

Faulty empty cells may not always hold the default value "0". If some input cells of a faulty empty cell change, it is possible that the faulty cell should hold a non-zero value, causing an error. For example, as shown in Fig. 1, we have recognized that the value of cell G11 should be calculated by adding cells B11, C11 and E11. Thus, cell G11 should hold the value 0.5, and contains a wrong value.

Note that, harmless empty cells usually do not degrade the quality of spreadsheets. Therefore, in this paper, we focus on faulty empty cells in which some formulas should be filled.

B. EmptyCheck Overview

Detecting faulty empty cells in spreadsheets needs to address two technical challenges. Let us explain them using the example shown in Fig. 1. First, which empty cells are faulty? How to detect them? Many empty cells exist because they are necessary to design the layout of spreadsheets, e.g., A2 and C3. We must filter out them to avoid producing too many false positives. Second, how to confirm the harmfulness degree of faulty empty cells? The faulty empty cells that have caused errors are more serious than other faulty empty cells.

	A	B	C	D	E	F	G
1	GRADES 9-12	1999 PARCEL TAX PROGRAM			1987 PARCEL TAX PROGRAM		TOTAL OF BOTH
2		FTE		COST	FTE	COST	PARCEL TAX
3		New	Continuing	2001-02	Continuing	2001-02	PROGRAMS
4	Science		0.6	48653	**0.8**	66991	=B5+C5+E5
5	Math		0.4	28812	0.4	32982	=B6+C6+E6
6	Music/Performi	0.6	1.6	145056	1	73552	=B7+C7+E7
7	Performing			3000		5000	
8	Technical/Voca				**1.4**	108784	=B9+C9+E9
9	Athletics		0.4	33175			=B10+C10+E10
10	AHS Athletic				0.5	116761	
11	AHS/AMS			10000			
12							
13	Library	0.2	0.8	70084			=B14+C14
14	(Classified)		0.4	9986			=B15+C15
15	Counseling		2.6	221035			=B16+C16+E16
16	Foreign Langua		**0.4**	26466			=B17+C17+E17
17							
18	Writing				0.6	42030	=B19+C19+E19
19	TOTALS:	=SUM(B5:B19)	=SUM(C5:C19)	=SUM(D5:D19)	=SUM(E5:E19)	=SUM(F5:F19)	=SUM(G5:G19)

(s1) Extract tables

	A	B	C	D	E	F	G
1	GRADES 9-12	1999 PARCEL TAX PROGRAM			1987 PARCEL TAX PROGRAM		TOTAL OF BOTH
2		FTE		COST	FTE	COST	PARCEL TAX
3		New	Continuing	2001-02	Continuing	2001-02	PROGRAMS
4	Science		0.6	48653	**0.8**	66991	=B5+C5+E5
5	Math		0.4	28812	0.4	32982	=B6+C6+E6
6	Music/Performi	0.6	1.6	145056	1	73552	=B7+C7+E7
7	Performing			3000		5000	
8	Technical/Voca				**1.4**	108784	=B9+C9+E9
9	Athletics		0.4	33175			=B10+C10+E10
10	AHS Athletic				0.5	116761	
11	AHS/AMS			10000			
12							
13	Library	0.2	0.8	70084			=B14+C14
14	(Classified)		0.4	9986			=B15+C15
15	Counseling		2.6	221035			=B16+C16+E16
16	Foreign Langua		**0.4**	26466			=B17+C17+E17
17							
18	Writing				0.6	42030	=B19+C19+E19
19	TOTALS:	=SUM(B5:B19)	=SUM(C5:C19)	=SUM(D5:D19)	=SUM(E5:E19)	=SUM(F5:F19)	=SUM(G5:G19)

(s2) Identify cell arrays

	A	B	C	D	E	F	G
1	GRADES 9-12	1999 PARCEL TAX PROGRAM			1987 PARCEL TAX PROGRAM		TOTAL OF BOTH
2		FTE		COST	FTE	COST	PARCEL TAX
3		New	Continuing	2001-02	Continuing	2001-02	PROGRAMS
4	Science		0.6	48653	**0.8**	66991	=B5+C5+E5
5	Math		0.4	28812	0.4	32982	=B6+C6+E6
6	Music/Performi	0.6	1.6	145056	1	73552	=B7+C7+E7
7	Performing			3000		5000	
8	Technical/Voca				**1.4**	108784	=B9+C9+E9
9	Athletics		0.4	33175			=B10+C10+E10
10	AHS Athletic				0.5	116761	
11	AHS/AMS			10000			
12							
13	Library	0.2	0.8	70084			=B14+C14
14	(Classified)		0.4	9986			=B15+C15
15	Counseling		2.6	221035			=B16+C16+E16
16	Foreign Langua		**0.4**	26466			=B17+C17+E17
17							
18	Writing				0.6	42030	=B19+C19+E19
19	TOTALS:	=SUM(B5:B19)	=SUM(C5:C19)	=SUM(D5:D19)	=SUM(E5:E19)	=SUM(F5:F19)	=SUM(G5:G19)

(s3) Remove overlapping cells from cell arrays

	A	B	C	D	E	F	G
1	GRADES 9-12	1999 PARCEL TAX PROGRAM			1987 PARCEL TAX PROGRAM		TOTAL OF BOTH
2		FTE		COST	FTE	COST	PARCEL TAX
3		New	Continuing	2001-02	Continuing	2001-02	PROGRAMS
4	Science		0.6	48653	**0.8**	66991	=B5+C5+E5
5	Math		0.4	28812	0.4	32982	=B6+C6+E6
6	Music/Performi	0.6	1.6	145056	1	73552	=B7+C7+E7
7	Performing			3000		5000	
8	Technical/Voca				**1.4**	108784	=B9+C9+E9
9	Athletics		0.4	33175			=B10+C10+E10
10	AHS Athletic				0.5	116761	
11	AHS/AMS			10000			
12							
13	Library	0.2	0.8	70084			=B14+C14
14	(Classified)		0.4	9986			=B15+C15
15	Counseling		2.6	221035			=B16+C16+E16
16	Foreign Langua		**0.4**	26466			=B17+C17+E17
17							
18	Writing				0.6	42030	=B19+C19+E19
19	TOTALS:	=SUM(B5:B19)	=SUM(C5:C19)	=SUM(D5:D19)	=SUM(E5:E19)	=SUM(F5:F19)	=SUM(G5:G19)

An error caused by faulty empty cell
Candidate formula:
=B11+C11+E11

(s4) Detect faulty empty cells

Fig. 2. Overview of EmptyCheck.

It is better if we can distinguish them and allow users to focus on more serious faulty empty cells first.

We handle the first challenge by extracting cell arrays in spreadsheets, and focus on detecting faulty empty cells that are adjacent to cell arrays. A cell array, proposed by Dou et al. [6], is a consecutive range of cells in a row or column prescribing certain computational semantic, e.g., [G5:G7], and all its cells share the same semantics. It is possible that some adjacent empty cells should belong to the cell array. For example, we can see that G8 should be part of the cell array [G5:G7]. According to the definition of cell arrays, these empty cells in a cell array should share the same computation pattern with other non-empty cells. Thus, empty cells in a cell array are faulty.

To overcome the second challenge, we should know whether a faulty empty cell's input cells are empty, too. The key point is how to get the input cells for a faulty empty cell. We can get the information from the cell array's computational semantics. For example, G8 is part of cell array [G5:G7], thus, we can infer that its input cells should be B8, C8 and E8.

III. Approach

Given a spreadsheet, EmptyCheck analyzes the context of empty cells and detects faulty empty cells in it. Fig. 2 presents how EmptyCheck works. First, EmptyCheck heuristically identifies tables (Section III.A). Next, EmptyCheck extracts cell arrays in each table (III.B and III.C). Finally, EmptyCheck detects the faulty empty cells in the extracted cell arrays, and determines whether the faulty empty cells have caused errors (III.D).

A. *Table Extraction*

A table is usually a rectangular block of cells, including header cells and body cells. A table usually represents a standalone business task, e.g., cells [A1:G20] in Fig. 1 make up a table for calculating the parcel tax. In a table, its body cells usually are placed continuously and form a data area. For example, the body of the table [A1:A20] in Fig. 1 is [B5:G20], marked by the red dashed rectangle in (s1).

In practice, users may put several tables into one worksheet. To extract data areas in a worksheet, we first classify cells into different types, and then extract tables based cells' types. We adopt the cell classification strategy proposed by UCheck [17] to classify all the cells into four types. 1) Data cells. The contents in data cells are numerical values or some special strings that are usually used as data instead of labels (e.g., "-", "#N/A"). 2) Formula cells. The formula cells contain formulas. 3) Label cells. Label cells usually contain strings, and are used as table headers. 4) Empty cells.

Then, we use *fence* to denote a row / column that can be used as the borders of tables. For each fence, it contains at least one label cell, and other cells in it are all empty. For example, the first four rows in the Fig. 1 are considered as *fences*. We further use *fences* to separate a worksheet into multiple data areas, e.g., [B5:G20], marked by the rectangle with red dashed borders in Fig. 2(s1).

Note that, our above table extraction approach differs slightly from previous approaches used in [6][7][17]. We observe that it is common for users to insert one or more empty rows or columns into a table (e.g., rows 13 and 18 in Fig. 1), making it more readable. To avoid extracting incomplete data areas, we do not use empty rows / columns as *fences*. For example, rows 13 and 18 in Fig. 2(s1) are not considered as *fences*, and we can extract a data area [B5:G20], rather than three small data areas [B5:G12], [B14:G17] and [B20:G20]. Thus, we can extract big data areas. This also allows us to take as many as possible empty cells into consideration and analyze them in the following steps.

B. Cell Array Extraction

Before elaborating how we extract cell arrays, we first introduce the *formula pattern* used in our approach. For each cell array, all their cells follow the same computational semantics, and thus the formulas in it share the same formula pattern. For example, cell *X*20 in [B20:G20] is computed by the sum of cells [*X*5:*X*19], where *X* = {B, C, D, E, F, G}. The formula pattern of the formulas in [B20:G20] is "SUM(R[-15]C:R[-1]C)", which is specified in the R1C1 style[1].

Basically, we adopt the cell array detection algorithm in CACheck [7]. However, we adapt it to detect cell arrays containing empty cells. The basic idea of cell array detection algorithm is that if a group of *consecutive* cells in a row / column satisfy the following conditions, these cells can be treated as a cell array:

- The group contains only data, formula and empty cells. And, there are at least two cells in the group.
- The group contains at least one formula cell. The information of formulas is the key for us to identify faulty empty cells. If there are no formulas in a cell array, we cannot infer whether its empty cells should be filled with formulas or not.
- All formulas in the group should reference their input cells in the similar way. If two formulas have some common R1C1 expressions for their input cells, we consider they reference their input cells in the similar way. For example, the formulas in cells G5 and G14 in Fig. 2(s2), their formula are "RC[-5]+RC[-4]+RC[-2]" and "RC[-5]+RC[-4]" in the R1C1 style, respectively. We can see that these two formulas have some common R1C1 expressions for their input cells, i.e., RC[-5] and RC[-4].

For each row and column in a data area, our algorithm is the same. Here, we only use the cells in a column to describe our algorithm. Specifically, for each column in a data area, our cell array detection algorithm works as follows. We use [*x*, *y*] to denote the cells in the column, where *x* denotes the row index of the first cell, and *y* denotes the row index of the last cell. 1) Initially, the cells in [*x*, *y*] are treated as a potential cell array [*m*, *n*]. 2) If the potential cell array [*m*, *n*] contains no formula, our algorithm terminates. 3) If the potential cell array [*m*, *n*] satisfies the above three conditions, we have detected a cell array. We further construct a new potential cell array [*n*+1, *y*], and check this new potential cell array from step 2 recursively. 4) If the potential cell array [*m*, *n*] does not satisfy the above three conditions, we change the potential cell array to [*m*, *n*-1], and check this changed potential cell array from step 2 recursively. We call the cell arrays extracted from each row as row-based cell arrays, and cell arrays extracted from each column as column-based cell arrays.

Taking the column G in the data area [G5:G20] in Fig. 2 (s1), as an example. Initially, we treat cells [G5:G20] a potential cell array. [G5:G20] is not a cell array, because the formula in G20 does not reference its input cell in the similar way as cell G19. Thus, we remove the last cell in [G5:G20], and treat cells [G5:G19] as a new potential cell array. The formulas in [G5:G19] reference their input cells in the similar way, i.e., they share the common input R1C1 expressions RC[-5] and RC[-4]. Thus, [G5:G19] is detected as a cell array. Because there is only cell G20 left, the detection algorithm terminates for column G. Therefore, in column G, we detect a column-based cell array [G5:G19].

For the data area B5:G20 in Fig. 2(s1), we in total detect 11 row-based cell arrays ([B*i*:G*i*], $i = 5, 6, 7, 9, 10, 14 - 17, 19, 20$) and 6 column-based cell arrays ([G5:G19] and *y*5:*y*20, *y* = B, C, D, E, F). For example, Fig. 2(s2) only shows two row-based cell arrays ([B5:G5] and [B20:G20]) and one column-based cluster ([G5:G19]).

Note that, CACheck [7] uses empty cells as boundaries of cell arrays. As a result, all generated cell arrays do not contain empty cells, and empty cells can separate a cell array (e.g., [G5:G19]) into multiple smaller ones, e.g., [G5:G7], [G9:G10] and [G14:G17]. In EmptyCheck, we solve these problems by considering empty cells as part of a cell array. Thus, EmptyCheck can extract complete cell arrays, e.g., [G5:G19].

C. Remove Overlapping Cells from Cell Arrays

Since we extract cell arrays in both row and column directions, the row-based cell arrays and the common-based cell arrays may share some cells. As shown in Fig. 2(s2), the row-based cell array [B5:G5] and the column-based cell array [G5:G19] share the common cell G5. According to the information of the headers of two cell arrays, we can recognize that G5 should only belong to the column-based cell array [G5:G19].

Before explaining how we remove overlapping cells from cell arrays, we first introduce how we recover formula pattern for a cell array. In a cell array, all its formulas can be treated as potential formula patterns. We select the formula which can cover most non-empty cells as its formula pattern. If a cell is covered, its data can be computed by the formula pattern. For example, cell array [G5:G19] has two potential formula patterns: "RC[-5]+RC[-4]+RC[-2]" (the R1C1 form of formulas B*i*+C*i*+E*i*) and "RC[-5]+RC[-4]" (the R1C1 form of formulas B*i*+C*i*). The potential formula pattern

[1]In spreadsheets, cell references can be represented in two styles: A1 and R1C1. In the A1 style, a cell at the *x*-th row and *y*-th column is denoted as *yx* in relative reference (e.g., B2), and \$*y*\$*x* in absolute reference (e.g., \$B\$2). In the R1C1 style, a cell at *m* rows below and *n* columns right to the current cell is denoted as R[*m*]C[*n*] in relative reference, and a cell at the *m*-th row and *n*-th column is notated as R*m*C*n* in absolute reference.

"RC[-5]+RC[-4]+RC[-2]" can cover 10 non-empty cells, while, the potential formula pattern "RC[-5]+RC[-4]" can only cover 4 non-empty cells. Thus, we use "RC[-5]+RC[-4]+RC[-2]" as cell array [G5:G19]'s formula pattern. If multiple potential formula patterns can cover the same number of cells, we randomly select one as the cell array's formula pattern.

For an overlapping cell, we further judge which cell array it should belong to by analyzing how close the cell and the formula pattern of its corresponding cell array is. Our overlapping cell array removal algorithm works as follows:

- If the overlapping cell contains a formula, we count the number of the same formulas with the formula in overlapping cell (in R1C1 form) in each cell array and remove this cell from the cell array with few shared formulas. For cell arrays [B5:G5] and [G5:G19], the formula contained in the overlapping cell G5 is "RC[-5]+RC[-4]+RC[-2]". Cell array [G5:G19] contains 7 common formulas, while cell array [B5:G5] contains none. Thus, cell G5 is removed from cell array [B5:G5], then cell array [B5:G5] changes into [B5:F5]. If the numbers of the same formulas are the same, we decide that the cell belongs to the row-based cell array.
- If the overlapping cell is data cell or empty cell, we calculate its expected value by using all involved cell arrays' formula patterns and the overlapping cell belongs to the cell array with the smallest distance between its actual value and expected value. We assume that most data in spreadsheets are correct. That means the overlapping cell's value should be calculated by the formula pattern of the corresponding cell array which it belongs to. We calculate the distance between its actual value and expected value: |*real value – expected value*|. Thus, the overlapping cell belongs to the cell array whose distance is the smallest. If two cell arrays have the same distance, we decide that the overlapping cell belongs to the row-based cell array.

In the above process, some cell arrays may be changed or added, e.g., a cell array is divided into two cell arrays by a removed cell in it. We further check whether the changed / added cell arrays satisfy the three conditions shown in Section III.B. If a cell array does not satisfy the conditions any more, we remove it from our detection results. For example, the cell array [B5:F5] is removed because it contains no formulas.

Through the above algorithm, we remove ten row-based cell arrays (B*i*:G*i*, *i* = 5, 6, 7, 9, 10, 14-17, 19) and five column-based cell arrays (*y*5:*y*19, *y* = B, C, D, E, F). Finally, we extract two cell arrays, a row-based cell array [B20:G20], marked by green color, and a column-based cell array [G5:G19], marked by yellow color, as shown in Fig. 2(s3).

D. Faulty Empty Cell Detection

Our approach described above can detect cell arrays with empty cells, e.g., cell array [G5:G19]. For all empty cells in a cell array, since they lack formulas, we will consider them as faulty empty cells. However, not all empty cells are faulty. For example, row 13 is used as a separator, and thus cell G13 should not be considered faulty.

Algorithm 1. Faulty empty cell detection

Input: *cell arrays* containing empty cells
Output: *FeCells* (faulty empty cells) and *SeCells* (faulty empty cells that cause errors)

```
1:  FeCells = ∅;   // Initialize faulty empty cells
2:  SeCells = 0;   // Initialize faulty empty cells that cause errors
3:  For each array ∈ cell arrays
4:      fpattern= recoverFormulaPattern(array);
5:      For each empty cell in array
6:          references =getReferences(fpattern, empty cell);
7:          If references only contain data, formula or empty cells
            in the table
8:              Add empty cell into FeCells
9:              If the calculated value for empty cell is nonzero
10:                 Add empty cell into SeCells
11:             EndIf
12:         EndIf
13:     EndFor
14: EndFor
15: Return FeCells and SeCells
```

Suppressing false positives. We observe that in a column-based (row-based) cell array, their cells should share the row (column) headers of the same types. The types of headers can be empty, string or numerical sequence (e.g., the headers are numerical sequence, like 1, 2, 3). Each cell has two headers: one row header and one column header. The nearest label cell or data cell in a numerical sequence in the same row can be treated as its row header. Similarly, the nearest label cell or data cell in a numerical sequence in the same column can be treated as its column header. For each cell in a column-based (row-based) cell array, we identify whether its row (column) header shares the same type with that of other non-empty cells in the same cell array. For example, in column-based cell array [G5:G19], non-empty cell G5 and empty cell G8 share the row headers with the same string type ("Science" and "Performing", respectively). However, the row headers of cells G5 and G13 have different types ("Science" and "", respectively), since empty cell G13's row header is null. Thus, the empty cell G13 is considered as a false positive, and removed from our detection result.

Identifying faulty empty cells. Our faulty empty cell detection algorithm can be divided into the following three steps and the pseudo code is shown in Algorithm 1.

1) Extracting the formula pattern for each cell array in which there is at least an empty cell (Line 4). For each cell array, we first extract all formulas from it and express them in the R1C1 style as possible formula patterns. We assume that most of the formulas in the cell arrays are correct. Based on this assumption, we simply select the formula pattern that can cover most cells (as mentioned in III.C) as the formula pattern for the cell array.

2) Identifying the faulty empty cells (Lines 7-12). According to the definition of a cell array, all cells in it should share the same semantics. In other words, the empty cells

should be filled with the formula pattern. After filled the formula pattern, the empty cells are reported as faulty empty cells only when all referenced cells are data cell or empty cells and they belong to the same table with the cell array.

3) Identifying the faulty empty cells that cause errors (Lines 9-11). For each faulty empty cell, we calculate its value according to the formula pattern and the values of corresponding input cells. The faulty empty cells are believed to cause errors when they get non-zero outputs.

Taking the column-based cell array in column G, marked by yellow color in Fig. 2(s3), as an example. As mentioned in Section III.C, we can get its formula pattern: "RC[-5]+RC[-4]+RC[-2]". The empty cell G8 will reference to B8, C8 and E8, after filled with formula "RC[-5]+RC[-4]+RC[-2]". All referenced cells are data cells. Thus, G8 is detected as faulty empty cell. Similarly, G11 and G12 are also detected as faulty empty cells. Further, we calculate the values for those three detected empty cells. We can see that cell G11 has non-zero output (0.5). Thus, G11 is considered as the faulty empty cell that caused an error.

IV. IMPLEMENTATION

In this section, we give a brief introduction about the implementation of EmptyCheck. For ease of use, we developed EmptyCheck as an excel plugin by using the Microsoft Visual Studio 2010 Tools for Office Language Pack (Version 4.0 Runtime) [18].

For visualization, EmptyCheck uses a widely used method, colors and tips, to visualize the detected faulty empty cells in spreadsheets: the faulty empty cells that have caused errors are marked by red color and others are marked by yellow color. The tips give the recommended formulas that should be filled. Fig. 2(s4) shows the visualized result of EmptyCheck. In this example, EmptyCheck detects three faulty empty cells (i.e., G8, G11 and G12) and one of them (G11, marked by red color) caused an error. When users click a faulty empty cell, the tips give the suggestion for the corresponding cell.

V. EVALUATION

Our evaluation studies the following three research questions:

RQ1: How common are faulty empty cells in real-life spreadsheets?

RQ2: Can EmptyCheck detect faulty empty cells precisely? Specifically, what are the precision, recall and F-Measure?

RQ3: How is EmptyCheck compared with existing techniques, e.g., the neighbor-based approach?

To answer the above three research questions, we evaluated EmptyCheck on 100 spreadsheets randomly selected from the EUSES corpus [15]. To answer RQ1, we manually checked all empty cells in these spreadsheets to determine whether they are faulty. To answer RQ2, we manually validated whether the faulty empty cells detected by EmptyCheck are true or not. To answer RQ3, we compared EmptyCheck with the neighbor-based approach [16].

TABLE I. STATISTICS OF 100 SELECTED SPREADSHEETS FROM THE EUSES CORPUS [15].

Categories	SS	WS	Formula	Empty
cs101	2	2	52	20
database	10	31	1,282	1,131
financial	14	19	1,332	1,948
forms3	2	2	95	50
grades	23	37	5,074	6,000
homework	24	31	3,001	2,647
inventory	17	58	2,612	4,093
modeling	8	10	2,851	3,190
Total	100	190	16,299	19,079

A. Experimental Subjects

We use the spreadsheets in the EUSES corpus [15] as our experimental subjects, which are the most frequently used spreadsheet corpus for spreadsheet-related studies. The EUSES corpus contains more than 4,000 spreadsheets. A previous study by Bas Jansen [19] shows that the spreadsheets in the EUSES corpus are similar to that in the Enron corpus [20], which are used by the Enron Corporation [21]. Thus, the spreadsheets in EUSES can represent the spreadsheets used in real life. Since our approach relies on the formula information, we only focus on the spreadsheets that contain at least one formula. In total, there are 1,617 spreadsheets containing formulas in the EUSES corpus. It is time-consuming and impractical to build a ground truth for our experiments using all these 1,617 spreadsheets. Therefore, we randomly sample 100 spreadsheets from them and build the ground truth for these spreadsheets by validating all empty cells and manually determining whether they are faulty or not. This process took the first two authors about three weeks.

Our random sampling algorithm works as follow. First, we randomly sample a corresponding number of spreadsheet from each domain according to the percentage of the total spreadsheets it contains. Because that the spreadsheets in The EUSES corpus [15] are clustered into different domains and each domain's spreadsheets are grouped in an individual folder. That allows us to avoid missing some domains' spreadsheets. Second, we try to understand the functionality of each selected spreadsheet and delete the spreadsheets that we cannot fully understand according to the limited information (e.g., the semantics of headers). Because the creators of these spreadsheets are not available, we need to manually validate whether empty cells in selected spreadsheets are faulty or not. If we cannot fully understand the spreadsheets, we may misjudge some empty cells. When deleting some incomprehensible spreadsheets, we randomly select the same number of spreadsheets from remained spreadsheets in the same domain. Finally, we selected 100 spreadsheets in total.

Table I shows the statistics of our experimental subject. The distribution of spreadsheets in different domains is shown in column SS. There are 190 worksheets (WS) in our experimental subject, from which we identify 16,299 formulas (Formula). As mentioned before, many empty cells are used for data layout. We in total got 19,079 empty cells (Empty).

B. Evaluation Metrics

Let $ECell_{groundtruth}$ denote the faulty empty cells in the ground truth that consist of all faulty empty cells in the selected spreadsheets from EUSES [15], $ECell_{detected}$ denote the cells detected by EmptyCheck. The precision, recall and F-Measure used in our evaluation are calculated as follows:

$$precision = \frac{|ECell_{detected} \cap ECell_{groundtruth}|}{|ECell_{detected}|} \quad (1)$$

$$recall = \frac{|ECell_{detected} \cap ECell_{groundtruth}|}{|ECell_{groundtruth}|} \quad (2)$$

$$F - Measure = \frac{2 \times precision \times recall}{precision + recall} \quad (3)$$

Next section, we will discuss how we build the ground truth in detail.

C. RQ1: How Common Are Faulty Empty Cells?

Since the creators of the spreadsheets in the EUSES corpus [15] are not available, we cannot get the ground truth from the original authors. Therefore, we manually validated all empty cells and determined whether they are faulty or not. We carefully checked each empty cell in the 100 selected spreadsheets, and try to answer the following questions for each empty cell:

1) Should it contain a formula?
2) If yes, can we recover a formula for the empty cell?
3) If yes, it is detected as faulty empty cell, and should the input cells of the empty cell have non-zero value?
4) If yes, this faulty empty cell causes an error.

To avoid possible mistakes, each empty cell has been cross-checked by the first two authors of this paper. Table II shows our manually validated result. In total, from these 100 selected spreadsheets, we find 486 faulty empty cells (Faulty Empty Cells / Total). Furthermore, we found that 350 faulty empty cells have caused errors (Faulty Empty Cells / Error). We can see that 72.02% faulty empty cells have caused errors (% E/T). Thus, faulty empty cells are harmful in spreadsheets and it is important to detect the faulty empty cells.

Table II also indicates how common faulty empty cells are. We can see that 36.00% (36 out of 100) of the selected spreadsheets (SS) and 24.21% (46 out of 190) of the worksheets (WS) contain at least one faulty empty cell. We found faulty empty errors exists in the cells in the spreadsheets of all categories except categories form3. These numbers indicate that faulty empty cells are common in real-life spreadsheets.

Therefore, we can draw the following conclusion:

The faulty empty cells are common in real-life spreadsheets. Most (72.02%) of faulty empty cells are harmful and have caused errors in spreadsheets.

TABLE II. STATISTICS OF FAULTY EMPTY CELL IN 100 SELECTED SPREADSHEETS FROM THE EUSES CORPUS[15].

Categories	SS	WS	Faulty Empty Cells		
			Total	Error	% E/T
cs101	1	1	3	3	100.00%
database	5	5	29	11	37.93%
financial	3	3	24	24	100.00%
forms3	0	0	0	0	0.00%
grades	4	6	69	59	85.51%
homework	7	7	68	50	73.53%
inventory	10	16	91	56	61.54%
modeling	6	8	202	147	72.77%
Total	36	46	486	350	72.02%

We use our manual validated results as the estimation of the real ground truth to answer RQ2 and RQ3.

D. RQ2: Detection Quality of EmptyCheck

To evaluate the quality of EmptyCheck, we ran EmptyCheck on the 100 sampled spreadsheets. Table III shows the detection results reported by EmptyCheck. We can see that EmptyCheck detected 564 faulty empty cells (Faulty Empty Cells / Detected), and 423 faulty empty cells are confirmed as true positives (Faulty Empty Cells / True). Thus, the precision of EmptyCheck is 75.00% (Faulty Empty Cells / Precision) and the recall is 87.04% (Faulty Empty Cells / Recall).

We further investigate the precision and recall of EmptyCheck in detecting the serious faulty empty cell that have caused errors. As shown in Table III EmptyCheck report 405 faulty empty cells that had caused errors (Faulty Empty Cells that Have Caused Errors / Detected), and among of them, 296 are confirmed as true positives (Faulty Empty Cells that Have Caused Errors / True), Thus, EmptyCheck can achieve high precision (73.09%) and recall (84.57%) in detecting the faulty empty cells that had caused errors, too.

Therefore, we can draw the following conclusion:

EmptyCheck can detect faulty empty cells with high precision (75.00%) and recall (87.04%).

False positives of detected faulty empty cells. The differences between the value of column *Detected* and that of column *True* in Table III are the false positives. We inspect all 141 false positives, and find out the reasons why EmptyCheck cannot achieve higher precision. (1) EmptyCheck failed to extract irregular tables correctly. Our table extraction algorithm always tries to extract the rectangle area as tables. While some tables whose boundaries are marked by bold borders of cells are not regular rectangle area. For an irregular table, EmptyCheck extract the minimum rectangle area which covers this table, containing some unrelated empty cells. 16 (11.35%) false positives belong to this case. (2) EmptyCheck identified headers for some cells incorrectly. Some cells' row headers are fixed length numbers (e.g., "164834" and "033835"). EmptyCheck identified them as data cells. 4 (2.84%) false positives belong to this case. (3) EmptyCheck

TABLE III. FAULTY EMPTY CELLS DETECTED BY EMPTYCHECK ON SAMPLED SPREADSHEETS.

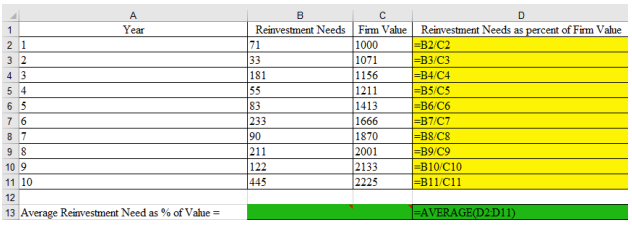
Categories	Faulty Empty Cell				Faulty Empty Cell that Have Caused Error			
	Detected	True	Precision	Recall	Detected	True	Precision	Recall
cs101	3	3	100.00%	100.00%	3	3	100.00%	100.00%
database	33	27	81.82%	93.10%	13	11	84.62%	100.00%
financial	48	23	47.92%	95.83%	44	23	52.27%	95.83%
forms3	0	0	0.00%	0.00%	0	0	0.00%	0.00%
grades	94	65	69.15%	94.20%	78	57	73.08%	96.61%
homework	85	68	80.00%	100.00%	63	50	79.37%	100.00%
inventory	119	56	47.06%	61.54%	76	33	43.42%	58.93%
modeling	182	181	99.45%	89.60%	128	119	92.97%	80.95%
Total	564	423	75.00%	87.04%	405	296	73.09%	84.57%

cannot understand the semantics of cells' headers. The semantics of the headers of cells usually determine which cells should contain formulas. Thus, our approach may misjudge situations and introduce false positives. 121 (85.82%) false positives belong to this case.

Fig. 3 shows an example of false positives caused by the third reason. This spreadsheet is used to process the data of reinvestment needs, containing four columns (*Year, Reinvestment Needs, Firm Value and Reinvestment Needs as percent of Firm Value*) and thirteen rows (rows 2-11 are the data of each year). EmptyCheck extracted a row-based cell array [B13:D13] (marked by green color) and a column-based cell array [D2:D11] (marked by yellow color). According to our faulty empty cells detection algorithm, EmptyCheck reports cells B13 and C13, because they should be filled in with formula "=AVERAGE(R[-11]C:R[-2]C)", as the same formula in D13. However, according to the row header of row 13, "*Average Reinvestment Need as % of Value =*", we can see that row 13 is used to calculate the average of reinvestment need as percent of firm value. Since all needed data is stored in column D, that only cell D13 is filled with formula is thus reasonable. In other words, cells B13 and C13 should be left to be empty.

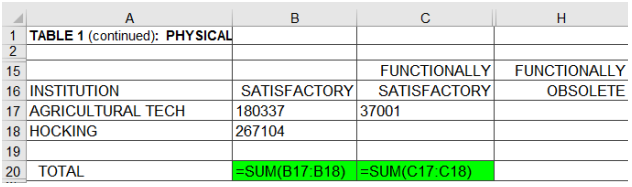
False negatives of detected faulty empty cells. We also inspected all 63 false negatives to find out the main reasons why EmptyCheck cannot achieve higher recall. (1) Incomplete table extraction. To avoid extracting incomplete data areas, we define a row / column that only contains at least a label cell and other cells in it are empty as a *fence*. We observed that some columns or rows contain no data in some tables, and they are identified as fence mistakenly. 37 (58.73%) false negatives belong to this case. (2) The headers of some faulty empty cells are missing. As mentioned before, EmptyCheck suppresses false positives according to the type information of headers. 22 (34.92%) true faulty empty cells are removed mistakenly. (3) EmptyCheck detects faulty empty cells by recovering the formulas for the empty cell. In some cases, EmptyCheck failed to recover correct formula according to the information contained in cell arrays. 4 (6.35%) false negatives belong to this case.

An example of false negatives caused by the first reason is shown in Fig. 4. This spreadsheet is used to handle the physical plant inventory according to the header in cell A1 ("*TABLE 1 (continued): PHYSICAL PLANT INVENTORY*").



	A	B	C	D
1	Year	Reinvestment Needs	Firm Value	Reinvestment Needs as percent of Firm Value
2	1	71	1000	=B2/C2
3	2	33	1071	=B3/C3
4	3	181	1156	=B4/C4
5	4	55	1211	=B5/C5
6	5	83	1413	=B6/C6
7	6	233	1666	=B7/C7
8	7	90	1870	=B8/C8
9	8	211	2001	=B9/C9
10	9	122	2133	=B10/C10
11	10	445	2225	=B11/C11
12				
13	Average Reinvestment Need as % of Value =			=AVERAGE(D2:D11)

Fig. 3. An example of false positives reported by EmptyCheck



	A	B	C	H
1	**TABLE 1** (continued): **PHYSICAL**			
2				
15			FUNCTIONALLY	FUNCTIONALLY
16	INSTITUTION	SATISFACTORY	SATISFACTORY	OBSOLETE
17	AGRICULTURAL TECH	180337	37001	
18	HOCKING	267104		
19				
20	TOTAL	=SUM(B17:B18)	=SUM(C17:C18)	

Fig. 4. An example of false negatives reported by EmptyCheck

This worksheet contains two tables A6:H10 and A15:H20. According to the header in A20 (*TOTAL*) and the formulas in cells B20 and C20, we can infer that cell H20 should be filled in with the formula "=SUM(R[-3]C:R[-2]C)", to calculate the total value of "*FUNCTIONALLY, OBSOLETE*". While, according our definition of fences, the column H is identified as fence to extract table. Thus, H20 is not reported as faulty empty cells.

E. RQ3: Comparison with Existing Techinques

Cunha et al. [16][22] proposed a neighbor-based approach to detect faulty empty cells: if an empty cell's four neighbor cells in the same row or column are all non-empty, then they consider it as a faulty empty cell. For example, for empty cell E8 in Fig. 1, its four neighbor cells are E5, E6, E7 and E9. Since these four cells are not empty, so E8 is considered as a faulty empty cell. However, that is wrong. So, the neighbor-based approach can introduce false positives. For another empty cell G12, since G11 is empty, so G12 is not considered as faulty empty cells. Thus, the neighbor-based approach may introduce false negatives, too.

We compared EmptyCheck and the neighbor-based approach on the same 100 sampled EUSES [15] spreadsheets. We inspected whether EmptyCheck can outperform existing techniques in detecting faulty empty cells. We did not compare EmptyCheck with other fault detection techniques, e.g., AmCheck [6], CACheck [7] and CUSTODES [8],

because these techniques are not designed to detect faulty empty cells and none of empty cells can be detected by them.

Table IV shows the detection result of the neighbor-based approach. We can see that the neighbor-based approach detected 334 faulty empty cells (Detected). However, only 18 of them are confirmed to be true positives (True). The precision and recall of the neighbor-based approach are 5.39% and 3.70%, respectively. While, EmptyCheck can achieve higher precision (75.00%) and recall (87.04%). The F-*Measure* of EmptyCheck is also much higher than that of neighbor-based approach (0.8057 vs 0.0439). Thus, EmptyCheck improves the state of the art greatly.

We further compare the detected faulty empty cells by two approaches. We find that EmptyCheck can detect all the faulty empty cells that were detected by the neighbor-based approach. While the neighbor-based method can only detect 4.26% of faulty empty cells detected by EmptyCheck. This indicates that EmptyCheck can detect much more faulty empty cells than the neighbor-based approach.

Therefore, we can draw the following conclusion:

EmptyCheck perform much better than existing techniques in detecting faulty empty cells.

TABLE IV. FAULTY EMPTY CELLS DETECTED BY THE NEIGHBOR-BASED APPROACH [16].

Category	Faulty Empty Cells			
	Detected	True	Precision	Recall
cs101	0	0	0.00%	0.00%
database	22	0	0.00%	0.00%
financial	21	0	0.00%	0.00%
forms3	0	0	0.00%	0.00%
grades	85	0	0.00%	0.00%
homework	98	0	0.00%	0.00%
inventory	10	0	0.00%	0.00%
modeling	98	18	18.37%	8.96%
Total	334	18	5.39%	3.70%

VI. DISCUSSION

A. Limitations

Although our experimental results indicate that EmptyCheck can detect faulty empty cells in spreadsheets with high precision and recall, it has some limitations.

EmptyCheck detect faulty empty cells by clustering empty cell into cell arrays. There exist some empty cells that are not adjacent to any cell arrays. We need to understand the concrete semantics (e.g., the table structure and header semantics) to know whether the empty cells are faulty or not (an example is discussed in Section V.D). It is very challenging to understand the semantics of headers, and thus detecting faulty empty cells in this case is also challenging. We leave this as future work.

To recover the formula pattern, EmptyCheck simply selects the formula which can cover most non-empty cells in each cell array based on the assumption that most of the formulas and data in the cell arrays are correct. This assumption is not always true. That means EmptyCheck may recommend wrong formulas for detected faulty empty cells. To alleviate this situation, the formula pattern synthesis algorithm designed in CACheck [7] can be employed.

B. Reference to Empty Cell

Reference to empty cells is studied by Cunha [22]. In their work, if a formula references an empty cell, it will be considered as a typical error. They propose an approach to detect reference to empty cells by checking every input cells. For example, cells B9 and C9 in Fig. 1 are empty and referenced by the formula in cell G9. Thus, cells G9 are considered to be smelly. Fixing the faulty empty cells detected by EmptyCheck may introduce reference to empty cell errors. For example, for cell G12, EmptyCheck detects it as faulty empty cell and advises users to fix this faulty empty cell by filling in with the formula "B12+C12+E12". After users fix this faulty cell, a reference to empty cell error will be introduced in G12, because Cells B12, C12 and E12 are empty cells. However, we believe that EmptyCheck is still important for spreadsheet quality improvement. First, these data cells are mainly left empty because there is no available input data. It is common to use empty cells to represent the default value "0" in spreadsheets. Second, EmptyCheck focuses on the empty cells that should contain formulas. They are caused by deleting formulas or forgetting to add formulas unintentionally. Thus, faulty empty cells detected by EmptyCheck can cause severe consequences.

VII. THREATS TO VALIDITY

Our experimental results indicate EmptyCheck can perform well in detecting faulty empty cells. We discuss some threats to our approach and evaluation in this section.

Representativeness of our experimental subject. The representativeness of our experimental subject is one threat to the validation of our evaluation. We select EUSES [15] as our experimental subject. It is because EUSES has been widely used in many spreadsheet-related studies. The spreadsheets in EUSES were extracted from World Wide Web, involving many different domains. A recent study carried out by Bas Jansen [19] also shows the spreadsheets in EUSES can represent the spreadsheets used in real life.

Manual validation of faulty empty cells. Since the creators of the spreadsheets in the EUSES corpus [15] are not available, we manually identified and validated whether empty cells are faulty or not. For our built ground truth, we cannot make sure that it does not contain any false positives or false negatives. We take two measures to alleviate possible mistakes. First, to ensure that we can understand the contents of spreadsheets, we avoid selecting overly complex spreadsheets. Second, every result is cross-checked by the first two authors of this paper.

Ground truth used in the experiments. Since it is impractical to obtain all faulty empty cells in the selected spreadsheets, we build the ground truth by checking all empty cells in them manually. This ground truth may contain some false positives or false negatives although we have done our best to avoid that. In the future, we will try to get a complete

ground truth in a small set of spreadsheets for which we can find the original authors.

VIII. Related Work

In this section, we introduce several pieces of related work concerning empty cell smell, spreadsheet fault / smell detection, testing and debugging.

Fault categorization. Cunha et al. [22] groups their spreadsheet smells into different categories: *Statistical Smells, Type Smells, Content Smells and Functional Dependencies Based Smells*. They also integrated the detection algorithms of these smells into a tools, SmellSheet Detective [16]. Hermans et al. [9][10] adapted code smells in the field of software engineering to spreadsheets. EmptyCheck proposes a new spreadsheet smell, faulty empty cell, as a subclass of *Type Smells*.

Spreadsheet fault / smell detection. AmCheck [6] and CACheck [7] are most related to our work. They firstly detect cell arrays, in which the cells should share the same semantics, and then repair the inconsistent formulas by synthesizing the correct formulas. However, cell arrays used in AmCheck and CACheck must be continuous and contain no empty cells. Thus, they cannot detect any faulty empty cell. UCheck [17] proposed a unit inference system and reports an error when its unit inference system cannot infer a unit in normal form for a cell. Dimension [23] is a reasoning system, which can infer dimension information to check the consistency of spreadsheet formulas. UCheck and Dimension rely on the formulas in the spreadsheets, thus they cannot detect faulty empty cells, which do not have any formulas. TableCheck [12] can detect table clones, in which the corresponding cells share the same header information, then detect related smells by identifying the inconsistencies among table clones. Hermans et al. [11] proposed a fingerprints based algorithm to detect exact and near-miss clones. It can facilitate finding and removing data clones However, spreadsheet clone detection techniques rely on that there are two similar areas in spreadsheets. EmptyCheck can detect faulty empty cells in single table. Cunha et al. [22] proposed that, if a formula references empty cells, it should be considered smelly. In summary, EmptyCheck is orthogonal to existing work.

Testing and debugging. WYSIWYT [4] was developed for testing spreadsheets. It provides the immediate feedback of the spreadsheet's tested-ness. AutoTest [5] can automatically generate test cases to help users test their spreadsheets. GoalDebug [14] allows users to specify the expected outputs for any concerned cells with incorrect outputs, then GoalDebug generates correction suggestions. CheckCell [13] is the first data debugging tool for spreadsheets, which can identify the cells that have high impact on the results of computation. The fault localization techniques attempt to help end-users locate fault by reducing the search space and prioritizing the sequence of the search through space [24][25][26][27]. Spectrum-based fault localization (SFL) [25][28] achieves localization process by ranking cells by their suspiciousness to contain a fault. Spectrum-enhanced dynamic slicing (SENDYS) [26] a technique that combines SFL with slicing-hitting-set-computation (SHSC) [24]. Constraint-based debugging [27] converts the formulas into a set of constraints. Those techniques rely on the contents in the cells, and the faulty empty cells cannot be detected and tested by them.

Spreadsheet evolution. VEnron [29] is the first versioned spreadsheet corpus, containing 360 evolution groups and 7,209 spreadsheets. SpreadCluster [30] is a similarity-based algorithm designed to identify different versions of the same spreadsheets. Hermans et al. [31] studied the spreadsheet evolution based on 54 pairs of spreadsheets. Each pair consists of a "bad" spreadsheet created by users and a "good" spreadsheet refactored by F1F9 for "bad" one. SheetDiff [32] can identify the differences between two versions of spreadsheet efficiently and effectively, providing more readable high-level changes. There are many spreadsheet comparison tools, such as DiffEngineX [33] and Synkronizer [34], aiming at detecting and visualizing differences between two spreadsheets. Faulty empty cells may be introduced during spreadsheet evolution, EmptyCheck can be used to detect the faulty empty cells in different versions of spreadsheets.

IX. Conclusion

Although some cells in spreadsheets are empty, they should have formulas according to their context. We find that this kind of faulty empty cells are common (36%) in real-life spreadsheets. In this paper, we propose a cluster-based approach, *EmptyCheck*, which can detect faulty empty cells automatically. Our evaluation on real-life spreadsheets in the EUSES corpus [15] shows that the faulty empty cells are common in real life spreadsheets. Most (72.02%) of faulty empty cells are harmful and have caused errors in spreadsheets. EmptyCheck can detect faulty empty cells with high precision and recall. While, existing techniques (e.g., the neighbor-based approach) can only detect 4.26% of faulty empty cells that are detected by EmptyCheck.

We plan to pursue our future work in three ways. 1) We can further improve the precision of EmptyCheck by taking the semantics of cell headers into consideration. 2) To improve the recall of EmptyCheck, we will explore more efficient table extraction and cell array detection algorithms. 3) We will explore more efficient approach, which are not able to be clustered into cell arrays, e.g., we can extract the context features of empty cells and detect faulty empty cells based on supervised machine learning.

Acknowledgment

This work was supported in part by National Natural Science Foundation of China (61702490, 61502011, 6171014), Beijing Natural Science Foundation (4164104), National Key Research and Development Plan (2016YFB1000803), Frontier Science Project of Chinese Academy of Sciences (QYZDJ-SSW-JSC036), Youth Innovation Promotion Association at CAS, and Beijing College Students' Research Project of High-Level Cross Cultivation of Undergraduate.

REFERENCES

[1] R. Panko, "Facing the Problem of Spreadsheet Errors," *Decis. Line*, vol. 35, pp. 8–10, 2006.

[2] C. Scaffidi, M. Shaw, and B. Myers, "Estimating the Numbers of End Users and End User Programmers," in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2005, pp. 207–214.

[3] Ko A J, Abraham R, Beckwith L, et al., "The State of the Art in End-User Software Engineering," *ACM Comput. Surv.*, vol. 43, pp. 1–44, 2011.

[4] G. Rothermel, L. Li, C. DuPuis, and M. Burnett, "What You See Is What You Test: A Methodology for Testing Form-Based Visual Programs," in *Proceedings of International Conference on Software Engineering (ICSE)*, 1998, pp. 198–207.

[5] R. Abraham and M. Erwig, "AutoTest: A Tool for Automatic Test Case Generation in Spreadsheets," in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2006, pp. 43–50.

[6] W. Dou, S.-C. Cheung, and J. Wei, "Is Spreadsheet Ambiguity Harmful? Detecting and Repairing Spreadsheet Smells due to Ambiguous Computation," in *Proceedings of International Conference on Software Engineering (ICSE)*, 2014, pp. 848–858.

[7] W. Dou, C. Xu, S. C. Cheung, and J. Wei, "CACheck: Detecting and Repairing Cell Arrays in Spreadsheets," *Trans. Softw. Eng.*, vol. 43, pp. 226–251, 2017.

[8] S.-C. Cheung, W. Chen, Y. Liu, and C. Xu, "CUSTODES: Automatic Spreadsheet Cell Clustering and Smell Detection using Strong and Weak Features," in *Proceedings of International Conference on Software Engineering (ICSE)*, 2016, pp. 464–475.

[9] F. Hermans, M. Pinzger, and A. van Deursen, "Detecting and Visualizing Inter-Worksheet Smells in Spreadsheets," in *Proceedings of International Conference on Software Engineering (ICSE)*, 2012, pp. 441–451.

[10] F. Hermans, M. Pinzger, and A. van Deursen, "Detecting and Refactoring Code Smells in Spreadsheet Formulas," *Empir. Softw. Eng.*, vol. 20, pp. 549–575, 2015.

[11] F. Hermans, B. Sedee, M. Pinzger, and A. van Deursen, "Data Clone Detection and Visualization in Spreadsheets," in *Proceedings of International Conference on Software Engineering (ICSE)*, 2013, pp. 292–301.

[12] W. Dou, S.-C. Cheung, C. Gao, C. Xu, L. Xu, and J. Wei, "Detecting Table Clones and Smells in Spreadsheets," in *Proceedings of ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE)*, 2016, pp. 787–798.

[13] D. W. Barowy, D. Gochev, and E. D. Berger, "CheckCell: Data Debugging for Spreadsheets," in *Proceedings of ACM International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA)*, 2014, pp. 507–523.

[14] R. Abraham and M. Erwig, "GoalDebug: A Spreadsheet Debugger for End Users," in *Proceedings of International Conference on Software Engineering (ICSE)*, 2007, pp. 251–260.

[15] M. Fisher and G. Rothermel, "The EUSES Spreadsheet Corpus: A Shared Resource for Supporting Experimentation with Spreadsheet Dependability Mechanisms," *ACM SIGSOFT Softw. Eng. Notes*, pp. 1–5, 2005.

[16] J. Cunha, J. P. Fernandes, P. Martins, J. Mendes, and J. Saraiva, "SmellSheet Detective: A Tool for Detecting Bad Smells in Spreadsheets," in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2012, pp. 243–244.

[17] R. Abraham and M. Erwig, "UCheck: A Spreadsheet Type Checker for End Users," *J. Vis. Lang. Comput.*, vol. 18, pp. 71–95, 2007.

[18] "Microsoft Visual Studio 2010 Tools for Office Language Pack (Version 4.0 Runtime)." [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=54246.

[19] B. Jansen, "Enron versus EUSES: A Comparison of Two Spreadsheet Corpora," in *Proceedings of Workshop on Software Engineering Methods in Spreadsheets (SEMS)*, 2015, pp. 41–47.

[20] F. Hermans and E. Murphy-Hill, "Enron's Spreadsheets and Related Emails: A Dataset and Analysis," in *Proceedings of the 37th IEEE International Conference on Software Engineering (ICSE)*, 2015, pp. 7–16.

[21] "Enron Corporation." [Online]. Available: https://en.wikipedia.org/wiki/Enron.

[22] J. Cunha, J. P. Fernandes, H. Ribeiro, and J. Saraiva, "Towards a Catalog of Spreadsheet Smells," *Lect. Notes Comput. Sci.*, vol. 7336, pp. 202–216, 2012.

[23] C. Chambers and M. Erwig, "Automatic Detection of Dimension Errors in Spreadsheets," *J. Vis. Lang. Comput.*, vol. 20, pp. 269–283, 2009.

[24] B. Hofer, A. Perez, R. Abreu, and F. Wotawa, "On the Empirical Evaluation of Fault Localization Techniques for Spreadsheets," *Autom. Softw. Eng.*, vol. 22, pp. 47–74, 2015.

[25] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "On the Accuracy of Spectrum-based Fault Localization," in *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION (TAICPART-MUTATION)*, 2007, pp. 89–98.

[26] B. Hofer, F. Wotawa, B. Hofer, and F. Wotawa, "Spectrum Enhanced Dynamic Slicing for Fault Localization," in *Proceedings of European Conference on Artificial Intelligence (ECAI)*, 2012, pp. 420–425.

[27] R. Abreu, A. Riboira, and F. Wotawa, "Constraint-based Debugging of Spreadsheets," in *Proceedings of Ibero-American Conference on Software Engineering*, 2012.

[28] E. Getzner, B. Hofer, and F. Wotawa, "Improving Spectrum-Based Fault Localization for Spreadsheet Debugging," in *proceedings of IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2017, pp. 102–113.

[29] W. Dou, L. Xu, S.-C. Cheung, C. Gao, J. Wei, and T. Huang, "VEnron: A Versioned Spreadsheet Corpus and Related Evolution Analysis," in *Proceedings of International Conference on Software Engineering Companion (ICSE)*, 2016, pp. 162–171.

[30] L. Xu, W. Dou, C. Gao, J. Wang, J. Wei, H. Zhong, T. Huang, "SpreadCluster: Recovering Versioned Spreadsheets through Similarity-Based Clustering," in *Proceedings of International Conference on Mining Software Repositories (MSR)*, 2017, pp. 158–169.

[31] B. Jansen and F. Hermans, "Code Smells in Spreadsheet Formulas Revisited on an Industrial Dataset," in *Proceedings of IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 372–380.

[32] C. Chambers, M. Erwig, and M. Luckey, "SheetDiff: A Tool for Identifying Changes in Spreadsheets," in *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2010, pp. 85–92.

[33] "Florencesoft DiffEngineX - Compare Excel Workbooks xlsx." [Online]. Available: https://www.florencesoft.com/.

[34] "Synkronizer Excel Compare: Compare, update and merge Excel files." [Online]. Available: http://www.synkronizer.com/.