Qualitative deviation models vs. quantitative models for fault localization in spreadsheets

Birgit Hofer, Iulia Nica, and Franz Wotawa[*]
Graz University of Technology, Institute for Software Technology,
Inffeldgasse 16b/2, 8010 Graz, Austria
e-mail: {bhofer,inica,wotawa}@ist.tugraz.at

Abstract

Automatizing fault localization in programs has been an interesting and active research field for several decades. In order to increase efficiency at least tool support for debugging would be highly required. This holds even more for end-user programs like spreadsheets. In this paper, we discuss three different models for spreadsheet debugging where one relies on qualitative algebra and deviations between the computed and the expected values of spreadsheet cells. The main objective of this paper is to clarify the questions whether a qualitative model for debugging can replace a model capturing the semantics of a spreadsheet. In order to answer this question, we carried out an empirical evaluation based on a publicly available spreadsheet corpus showing that qualitative models allow for computing similar diagnoses but require a fraction of runtime. Hence, qualitative models have the potential of being used for fault localization under real-time conditions.

Introduction

Debugging, i.e., the localization of a fault causing a misbehavior and its repair, is a taxing task even for professional programmers. The situation seems to be even worse when it comes to end-user programming where the creation of spreadsheets is one of the best-known examples. Panko (2008) mentioned that there is a 3-5 % human error rate when writing formulas in spreadsheets. In addition, there is almost no debugging support available in today's spreadsheet tools, besides simple mechanisms that allow indicating maybe unwanted patterns in equations. Hence, there is a need for providing debugging support for spreadsheet programming and research in this area is highly required.

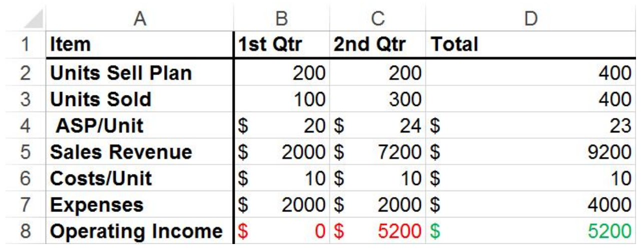
In this paper, we follow previous research of Wotawa (2016) dealing with the development of qualitative deviation models for diagnosis. In (Wotawa 2016), the author motivated research using an example from the spreadsheet domain, introduced different models, and provided first experimental results. In this paper, we want to answer the question about the effectiveness of various types of models for fault localization in spreadsheets. For this purpose, we took example spreadsheet programs used for evaluating debugging tools for spreadsheets and applied model-based diagnosis using different types of models. In particular, we compare the outcome of a diagnosis engine using a model that captures the behavior of a spreadsheet directly (i.e., the value-based model), a model that relies on data dependencies between references used in the spreadsheet (i.e., the dependency-based model), and a model that uses a qualitative algebra comprising $<$, $=$, and $>$ as values for handling functions used in spreadsheets (i.e., the comparison-based model). In the latter model the values $<$, $=$, and $>$ capture the case where a computed value is smaller, equivalent, or larger than the expected one.

We illustrate our approach by means of the spreadsheet given in Figure 1. This spreadsheet is a simplified and slightly modified version of the bugetone spreadsheet from the EUSES spreadsheet corpus (Fisher and Rothermel 2005). For demonstration purposes, we have inserted faults in the cells B7 and C7. These cells reference the cells B2 and C2 instead of B3 and C3. Because of these faults, the operating income of the first and the second quarter are erroneous. Coincidentally, the faults cancel each other when their computed values are summed up in cell D7. As a consequence, the total operating income given in cell D8 is correct.

All used models cannot identify a single cell as root source of the observed misbehavior. The value-based and the comparison-based model compute five double-fault diagnoses as explanation: {B5,C5}, {B5,C7}, {C5,B7}, {B7,C7} and {B8,C8}. The dependency-based model computes only one double-fault diagnosis, namely {B8,C8}. This means that the dependency-based model fails to identify the real double fault whereas the comparison-based model shows equivalent performance like the value-based model. However, there is a huge difference in the computation time required for obtaining all double-fault diagnoses. Whereas the value-based model requires 537 milliseconds for computing the diagnoses, the dependency-based and the comparison-based model require only three to four milliseconds. This indicates that a qualitative model would be both capable of being precise in diagnosis and being fast in the diagnosis computation.

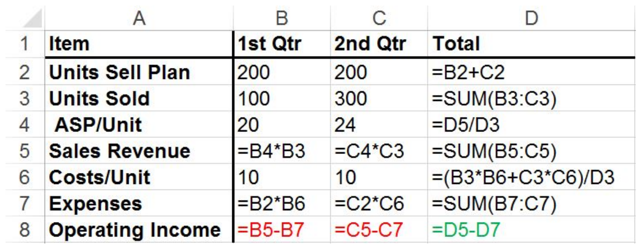
The main contribution of this paper is to present and discuss first empirical results of different types of models used for diagnosing spreadsheets. For the evaluation, we relied

[*]Authors are listed in alphabetical order.



	A	B	C	D
1	Item	1st Qtr	2nd Qtr	Total
2	Units Sell Plan	200	200	400
3	Units Sold	100	300	400
4	ASP/Unit	$ 20	$ 24	$ 23
5	Sales Revenue	$ 2000	$ 7200	$ 9200
6	Costs/Unit	$ 10	$ 10	$ 10
7	Expenses	$ 2000	$ 2000	$ 4000
8	Operating Income	$ 0	$ 5200	$ 5200

(a) Value view of the faulty spreadsheet



	A	B	C	D
1	Item	1st Qtr	2nd Qtr	Total
2	Units Sell Plan	200	200	=B2+C2
3	Units Sold	100	300	=SUM(B3:C3)
4	ASP/Unit	20	24	=D5/D3
5	Sales Revenue	=B4*B3	=C4*C3	=SUM(B5:C5)
6	Costs/Unit	10	10	=(B3*B6+C3*C6)/D3
7	Expenses	=B2*B6	=C2*C6	=SUM(B7:C7)
8	Operating Income	=B5-B7	=C5-C7	=D5-D7

(b) Formula view of the faulty spreadsheet

Figure 1: Running example. Correct output values are given in green color, erroneous output values are given in red color. There are two faulty cells: B7 and C7.

on artificial and also real-world spreadsheet examples from a corpus that has been often used for evaluating different kinds of debugging approaches.

We organize this paper as follows: First, we introduce the basic definitions where we define model-based diagnosis based on constraint solving. Afterwards, we discuss the different models to be compared and also define the mapping of spreadsheets to their constraint representation. Further on, we describe the empirical evaluation settings and the results obtained. Finally, we discuss related research and conclude the paper.

Basic definitions

In order to be self-contained, we briefly recall the basic definitions of model-based diagnosis relying on constraints taken from (Wotawa 2016): A constraint system is a tuple $(\mathit{VARS}, \mathit{DOM}, \mathit{CONS})$ where *VARS* is defined as a finite set of variables, *DOM* is a function mapping each variable to its domain comprising at least one element, and *CONS* is a finite set of constraints. Without restricting generality we define a constraint c as a pair $((v_1,\ldots,v_k), tl)$ where $(v_1,\ldots,v_k)$ is a tuple of variables from *VARS*, and tl a set of tuples $(x_1,\ldots,x_k)$ of values where for each $i \in \{1,\ldots,k\}$: $x_i \in \mathit{DOM}(v_i)$. The set of tuples tl in this definition declares all allowed variable value combinations for a particular constraint. For simplicity, we assume a function $\mathit{scope}(c)$ for a constraint c returning the tuple $(v_1,\ldots,v_k)$, and a similar function $tl(c)$ returning the set of tuples tl of c.

Using the definition of constraints we can define the constraint satisfaction problem. For this purpose, we introduce the assignment of values to variables. Given a constraint system $(\mathit{VARS}, \mathit{DOM}, \mathit{CONS})$, and variable $v \in \mathit{VARS}$, then $v = x$ with $x \in \mathit{DOM}(v)$ is a single assignment of a value x to the variable v. We further define a value assignment as a set of single assignments where there is at maximum one single assignment for each variable. A constraint c with scope $(v_1,\ldots,v_k)$ fulfills a value assignment $\{\ldots,v_1 = x_1,\ldots,v_k = x_k,\ldots\}$, if there exists a tuple $(x_1,\ldots,x_k)$ in the constraint $tl(c)$. Otherwise, we say that such a value assignment contradicts the constraint.

Example 1. *Cell B5 contains the formula $B3 \cdot B4$. We can model this by using the following constraint: $((B3, B4, B5), \{(a, b, a \cdot b) | a, b \in \mathbb{N}\})$.*

A constraint satisfaction problem for a given constraint system can be defined whether there exists a value assignment that fulfills all given constraints. If there is such a value assignment, the constraint satisfaction problem is said to be fulfilled. Solving a constraint satisfaction problem is basically a search procedure for a value assignment that fulfills all constraints. This search for constraint systems having only constraints with finite tuple lists is exponential and its corresponding problem is NP-complete. For details about algorithms and heuristics we refer the interested reader to (Dechter 2003).

In the following, we discuss the diagnosis problem and show how constraint solving can be used to solve the classical diagnosis problem. According to Reiter (1987), a diagnosis problem is a tuple $(\mathit{COMP}, \mathit{SD}, \mathit{OBS})$ where *COMP* is a set of components, *SD* a logical sentence describing the behavior of the system, i.e., the system description, and *OBS* a set of observations. In our constraint-based representation of the diagnosis problem, we assume a constraint representation of the system and additional constraints specifying the observations. The constraint representation of a diagnosis problem (or the diagnosis problem for short) is a tuple $(\mathit{VARS}, \mathit{DOM}, \mathit{CONS} \cup \mathit{COBS})$ where $(\mathit{VARS}, \mathit{DOM}, \mathit{CONS})$ is a constraint representation of a system comprising variables ab_C for every component C of the system, and *COBS* is the constraint representation of all observations *OBS*.

Example 2. *(Continuation of Example 1) Assuming that the values of B3 and B4 are correct, then B5 either computes the desired output or its formula is faulty. We can model this by using the following constraint: $((ab_{B5}, B3, B4, B5), \{(F, a, b, a \cdot b) | a, b \in \mathbb{N}\} \cup \{(T, a, b, c) | a, b, c \in \mathbb{N}\})$.*

We can create such constraints for all formula cells. This set of constraints builds the system description SD. The formula cells build the set of components COMP, i.e. $\mathit{COMP} = \{B5, B7, B8, C5, C7, C8, D2, \ldots, D8\}$. The set of observations contains the information about the correct and erroneous output as well as the input cells and their values: $\mathit{OBS} = \{B8 = 1000, C8 = 4200, D8 = 5200, B2 = 200, B3 = 100, \ldots\}$.

The constraint representation of this diagnosis problem is as follows: The set of variables comprises the input cells and the formula cells $(B1,\ldots,D8)$, some auxiliary variables for interim results $(t_1,\ldots,t_n)$ and an abnormal variable for each formula cell $(ab_{B5},\ldots ab_{D8})$: $\mathit{VARS} = \{B2,\ldots,D8, t_1,\ldots,t_n, ab_{B5},\ldots ab_{D8}\}$. The domain for the abnormal variables is Boolean, $(\forall(w) \in \{ab_{B5},\ldots ab_{D8}\} : \mathit{DOM}(w) = \{\mathit{true},\mathit{false}\})$; the domain of the other variables is Integer $(\forall(w) \in$

$\{B2, \ldots, D8, t_1, \ldots, t_n, \}\}$: *DOM*$(w) = \mathbb{N}$*). The set of constraints CONS = SD* $\cup$ *COBS where COBS is a constraint representation of OBS.*

The results of a diagnosis problem, i.e., the diagnoses, are subsets of the set of components *COMP*. We obtain these subsets from the solutions of the corresponding constraint problem via taking one value assignment that is a solution, and putting all components C for which the corresponding variable ab_C is set to T into a set, i.e., if s is a solution of the constraint representation of a diagnosis problem, then its corresponding diagnosis is $\Delta_s = \{C|(ab_C = T) \in s\}$. When computing all solutions from the constraint representation, we obtain all possible diagnoses. As usual, we define a diagnosis to be minimal if there exists no subset, which is itself a diagnosis.

We are interested in only computing minimal diagnoses in the most efficient way. In the following, we discuss a diagnosis algorithm that computes minimal diagnoses of increasing size. This can be achieved via restricting the number of ab_C variables to be set to true using constraints. In this way we are able to compute diagnoses up to a pre-specified size. The necessary additional constraints are added during diagnosis computation in diagnosis algorithms like ConDiag (see Algorithm 1). Nica and Wotawa (2012) introduced the ConDiag algorithm that computes minimal diagnoses up to a predefined size using a constraint representation of the diagnosis problem. Nica *et al.* (2013) compared ConDiag with other diagnosis algorithms showing a good overall runtime.

Algorithm 1 ConDiag$((\mathit{VARS}, \mathit{DOM}, \mathit{CONS} \cup \mathit{COBS}), \mathit{COMP}, n)$

Input: A constraint model $(\mathit{VARS}, \mathit{DOM}, \mathit{CONS} \cup \mathit{COBS})$ of a system having components *COMP* and the desired diagnosis cardinality n
Output: All minimal diagnoses up to the predefined cardinality n

```
 1: Let $DS$ be $\{\}$
 2: Let $M$ be $\mathit{CONS} \cup \mathit{COBS}$
 3: for $i = 0$ to $n$ do
 4:     $CM = M \cup \{|\{ab_C | C \in \mathit{COMP} \wedge ab_C = T\}| = i\}$
 5:     $S = \mathcal{P}\,(\mathbf{CSolver}(\mathit{VARS}, \mathit{DOM}, CM))$
 6:     if $i$ is 0 and $S$ is $\{\{\}\}$ then
 7:         return $S$
 8:     end if
 9:     Let $DS$ be $DS \cup S$.
10:     $M = M \cup \{\neg(\mathcal{C}(S))\}$
11: end for
12: return $DS$
```

The ConDiag algorithm computes all diagnoses starting with cardinality 0 to the predefined size n that has to be provided as parameter. In each step, we are searching for solutions that have exactly a size of i (Step 4). All these solutions are added to the set of solutions in Step 9. In order to prevent the computation of non-minimal diagnoses additional constraints saying that we are not interested in superset diagnoses are added (see Step 10). ConDiag returns all minimal diagnoses up to size n and the empty diagnosis if the system works as expected.

Example 3. *When calling* ConDiag *with the sets described in Example 2 and* $n = 3$*, we obtain 5 double fault diagnoses and 12 triple fault diagnoses: In the first and second iteration (i=0 and i=1), the solver cannot find any solution. In the third iteration (i=2), the solver returns the set* $\{\{$*B5,C5*$\}$, $\{$*B5,C7*$\}$, $\{$*C5,B7*$\}$, $\{$*B7,C7*$\}$ *and* $\{$*B8,C8*$\}\}$ *as solution. We now have to add blocking clauses* $(\neg(ab_{B5} = T \wedge ab_{C5} = T), \ldots)$ *for all found diagnoses to the set of constraints (see* ConDiag *Line 10). This prevents the solver from reporting supersets of these diagnoses as a solution. In the fourth iteration (i=3), the solver returns 12 solutions. If we would not block the double fault diagnoses, the solver would return 63 triple fault solutions.*

Models for diagnosis

For debugging spreadsheets we need constraint models. In particular, we are going to discuss the three different models discussed in (Wotawa 2016), (Hofer and Wotawa 2014) and (Hofer, Hoefler, and Wotawa 2017). The value-based model (V) relies on the concrete functions. The dependency-based model (D) makes use of dependencies among variables, and the comparison model (C) is a deviation model considering a comparison between the expected and the real behavior of a spreadsheet. For all three models, we briefly outline their basic principles and show how spreadsheets can be converted into them. Without restricting generality, we assume that a spreadsheet comprises a set of cells each uniquely identifiable using the row and the column. Rows are usually denoted using integer values and columns using combinations of letters alphabetically ordered, i.e., starting from A, B, ..., Z, AA, AB, ... Each cell may comprise a value or a function. In the latter case, the value of a cell is equivalent to the value computed using the function. For simplicity, we restrict ourselves to integer values and functions like $+$, $-$, $\star$, and $/$ having two arguments as inputs where the inputs can only be references to other cells. These restrictions to integers and simple arithmetical operators can be easily overcome but would require additional definition overhead.

Models are constraint systems comprising a set of variables *VARS*, a domain function *DOM* mapping a domain to each variable, and a set of constraints *CONS*. For each non-empty cell XY in a spreadsheet where X indicates the column and Y the row, we add a constraint variable v_{XY} to *VARS*, and define *DOM*(v_{XY}) accordingly to the type of model. In addition, we need abnormal variables for stating that some of the cells are correct or not correct. For this purpose, we add a constraint variable ab_{XY} to *VARS* for each non-empty cell XY storing a function. For such variables, *DOM*$(ab_{XY}$ is defined as being Boolean $\{T, F\}$ comprising true (T) and false (F) as values. The conversion of the cells lead to the set of constraints *CONS*. For all three models, this conversion is done for each non-empty cell separately considering the underlying modeling concept V, D, and C.

Let us first start with the value-based model V. In a *value-based model* V, the constraints corresponding to one cell

XY represent a value or function. Cell values have to be of domain INTEGER. Hence, we assume $\mathit{DOM}(v_{XY}) =$ INTEGER for each non-empty cell XY. Regarding the constraints, we do not add any constraints for cells storing only values. For a cell XY with a function of the form $op(x, y)$ where $op \in \{+, -, *, /\}$ is an operator and x, y are references, we add $((ab_{XY}, v_{XY}, v_x, v_y), \{(F, a, b, c)|a, b, c \in$ INTEGER$\wedge a = b\, op\, c\} \cup \{(T, a, b, c)|a, b, c \in$ INTEGER$\})$ to *CONS*. Because the INTEGER domain is finite, the number of value tuples of this constraint are also finite. For the experiments we restrict number values to even smaller domains.

Example 4. *The constraints described in Example 2 correspond to the value-based model V.*

For the *dependency-based model* D, we make use of the following underlying idea. Given a function $op(x, y)$ with references to cells x and y. If both cells x and y are correct, and assuming the equation to be correct, we are able to derive that the result of the function has to be correct. Accordingly to (Hofer, Hoefler, and Wotawa 2017), this model can be improved when assuming that masking of failures does not occur, meaning that there is no coincidental correctness and a correct return value of a function would also imply correct input values when assuming the function to be correct. When considering integer domains, failure masking only occurs in very specific circumstances, e.g., in case of multiplication where one input value is zero. For modeling using modeling concept D, we do no longer have INTEGER domains. Instead, we use the values ok and $\neg ok$ for stating that a certain value is correct or not correct respectively, i.e., $\mathit{DOM}(v_{XY}) = \{ok, \neg ok\}$. The modeling for a cell with the underlying model concept D is done as follows: (i) For a value x stored in cell XY, we do not add any constraint to *CONS*. (ii) For functions $op(x, y)$ with referenced cells x and y, we add the constraint $((v_{XY}, v_x, v_y), T_D)$ to *CONS* where T_D is a set comprising the following elements:

ab_{XY}	v_{XY}	v_x	v_y
F	ok	ok	ok
F	$\neg ok$	$\neg ok$	ok
F	$\neg ok$	ok	$\neg ok$
F	$\neg ok$	$\neg ok$	$\neg ok$
T	.	.	.

In the above table the "." entries in the last row stand for value ok or $\neg ok$. Hence, in case of a fault, every value tuple is possible to occur.

Example 5. *The dependency-based constraint representation of our running example is as follows:*

$$\mathit{VARS} = \{B2, \ldots, D8, ab_{B5}, \ldots, ab_{D8}\}$$

$$\forall (w) \in \{ab_{B5}, \ldots, ab_{D8}\} : \mathit{DOM}(w) = \{\mathit{true}, \mathit{false}\}$$

$$\forall (w) \in \{B2, \ldots, B8\}\} : \mathit{DOM}(w) = \{ok, \neg ok\}$$

$$\mathit{CONS} = \{((B5, B3, B4), T_D), ((B7, B2, B6), T_D), \ldots\}.$$

For the *comparison model* (C), we introduce a three value domain for each cell XY comprising <, =, and > indicating that the value of this cell is smaller, equivalent, or larger than the expected value. Hence, for all non-empty cells XY we define $\mathit{DOM}(XY) = \{<, =, >\}$. For the operators, we specifically define constraint tables. For example, if we add two numbers and one number is larger than expected and the other is equivalent, the outcome must also be larger. Similar rules can be obtained for other values and also operators. Similar to D, we do not have constraints for cells only storing values, and for functions $op(x, y)$ with referenced cells x and y, we add the constraint $((v_{XY}, v_x, v_y), T_i)$ to *CONS* where T_i is a set of tuples comprising the following elements for operators $*, +$ on the left (T_1) and $-, /$ on the right side (T_2) respectively:

ab_{XY}	v_{XY}	v_x	v_y
F	$=$	$=$	$=$
F	$<$	$<$	$=$
F	$<$	$=$	$<$
F	$<$	$<$	$<$
F	$>$	$>$	$=$
F	$>$	$=$	$>$
F	$>$	$>$	$>$
F	$=$	$<$	$>$
F	$<$	$<$	$>$
F	$>$	$<$	$>$
F	$=$	$>$	$<$
F	$<$	$>$	$<$
F	$>$	$>$	$<$
T	.	.	.

ab_{XY}	v_{XY}	v_x	v_y
F	$=$	$=$	$=$
F	$<$	$<$	$=$
F	$>$	$=$	$<$
F	$<$	$<$	$<$
F	$=$	$<$	$<$
F	$>$	$<$	$<$
F	$>$	$>$	$=$
F	$<$	$=$	$>$
F	$<$	$>$	$>$
F	$=$	$>$	$>$
F	$>$	$>$	$>$
F	$<$	$<$	$>$
F	$>$	$>$	$<$
T	.	.	.

Again, the "." indicates all values <, =, >.

Example 6. *The comparison-based constraint representation of our running example is as follows:*

$$\mathit{VARS} = \{B2, \ldots, D8, ab_{B5}, \ldots ab_{D8}\}$$

$$\forall (w) \in \{ab_{B5}, \ldots ab_{D8}\} : \mathit{DOM}(w) = \{\mathit{true}, \mathit{false}\}$$

$$\forall (w) \in \{B2, \ldots, B8\}\} : \mathit{DOM}(w) = \{<, =, >\}$$

$$\mathit{CONS} = \{((B5, B3, B4), T_1), \ldots, ((B8, B5, B7), T_2), \ldots\}.$$

In addition to the constraints representing the functionality of the cells, we have to add observations. In the case of spreadsheets, we have to add constraints for the non-empty cells comprising values only. Usually, input cells, i.e., cells that are used as references but that do not reference any cells, are assumed to have the expected values. Output cells, i.e., cells that are never used as references, may comprise expected or unexpected values. For the value-based model V, cells XY may comprise their values or any other expected value. For such non-empty cells storing a value x, we add $((v_{XY}), \{(x^e)\})$ to the constraint representation of the observations *COBS*. There x^e indicates the expected value. For the *dependency-based model* D, we either add $((v_{XY}), \{(ok)\})$ or $((v_{XY}), \{(\neg ok)\})$ to *COBS* depending whether the value stored in XY is correct or not respectively. For the *comparison model* C, we add $((v_{XY}), \{(<)\})$, $((v_{XY}), \{(=)\})$, or $((v_{XY}), \{(>)\})$ to *COBS* depending whether the value of cell XY is smaller, equivalent, or larger than the expected value.

Example 7. *For the value-based model V, the constraints of the observations are* $\mathit{COBS} = \{((B2, B3,$

$\ldots, B8, C8, D8), \{(200, 100, \ldots, 1000, 4200, 5200)\})\}$. *For the dependency-based model D, the constraints of the observations are COBS* $= \{((B2, B3, \ldots, B8, C8, D8), \{(ok, ok, \ldots \neg ok, \neg ok, ok)\})\}$. *For the comparison-based model C, the constraints of the observations are COBS* $= \{((B2, B3, \ldots, B8, C8, D8), \{(=, =, \ldots <, >, =)\})\}$.

Empirical Evaluation

This empirical evaluation compares the comparison-based model with the value-based model and the improved dependency-based model w.r.t. runtime and diagnostic accuracy. For solving the constraints, we used Minion (Gent, Jefferson, and Miguel 2006), an out-of-the-box, open source constraint solver that supports arithmetic, relational, and logic constraints over Boolean and Integers.

As evaluation basis, we used the Integer spreadsheet corpus (Ausserlechner et al. 2013)[1]. This corpus contains both, real life spreadsheets as well as artificially created spreadsheets. Each spreadsheet contains up to three artificially seeded faults. We used only a subset of spreadsheets because our prototype does not support the conversion of IF and MAX expressions to constraints. Please note that this is a limitation of the prototype, not a limitation of the approach. We additionally excluded 16 spreadsheets, because Minion did not solve the value-based constraint system within 20 minutes. Finally, 48 spreadsheets remained for evaluation. The evaluation was performed on an Intel Core processor (2.90 GHz) with 8 GB RAM and Windows 8 (Windows 8.1 Enterprise, 64 bit) as operating system. We used Apache POI (https://poi.apache.org/) for parsing the spreadsheets and Java for converting them into constraints. For solving the constraints, we used MINION version 1.8. We applied the principle of early termination during the evaluation, i. e., double faults were only computed when no single faults could be found, and triple faults were only computed when no double faults could be found.

Table 1 shows the results for the 48 faulty spreadsheets. The solving time is the time Minion requires to parse and solve the given constraint system. For diagnoses of cardinality 2 and 3, the solving time is the sum of the solving times required for computing the single, double, and triple faults. The indicated times are the arithmetic mean over 10 runs. Whenever we have indicated 0 as solving time, Minion required less than 0.5 ms and therefore returned 0 ms as solving time. The value-based model requires significantly more time than the dependency-based and comparison-based model. This can be explained with the different domain sizes: While the dependency-based model reasons only over Boolean values and the comparison-based model restricts the variables' domains of Minion to $\{<, =, >\}$ (representing $<$, $=$, and $>$ respectively), the value-based model reasons of Integer values ranging from -2,000 to 50,000. Surprisingly, the comparison-based model requires even less computation time than the dependency-based model on average. This can be explained by the different Minion constructs that were used: We used tables as reasoning mechanism in the comparison-based model while we used 'watched-or' for the dependency-based models. In future work, we are going to evaluate if the comparison-based model will still be faster when modeling the constraints in a different way or when using a different solver.

When investigating the diagnostic accuracy of the models, we see that for half of the spreadsheets (i.e. 24 spreadsheets), all models compute the same amount of diagnoses. For 20 spreadsheets, the comparison-based and the dependency-based models compute the same diagnoses, while the value-based model computes less diagnoses. For two spreadsheets, the value-based model computes more diagnoses than the other models, because the value-based model does not find any single fault diagnoses, but the other models do. The combination of cells to double fault diagnoses increases the amount of diagnoses. For example, the comparison-based and the dependency-based model give three single fault diagnoses ({D4},{D5}, and {D6}) for the spreadsheet arithmetics01_3_1; the value-based model gives supersets of these diagnoses ({D4,J3},{D5,J3},{D6,J3}) as well as some additional diagnoses (e.g.{F3,J3}}. For one spreadsheet, namely arithmetics01_2_2, the comparison-based model computes less diagnoses than the dependency-based model and the value-based model. For another spreadsheet (arithmetics02_2_1), the comparison-based model computes more diagnoses than the dependency-based model (90 vs. 7). The reason for this is that the dependency-based model identifies single fault diagnoses, while the comparison-based model cannot find single fault diagnoses, but double fault diagnoses which are combination of the single faults with other cells.

When using the value-based model, 16 out of 22 double faults have been identified as single faults (i.e. one of the two faulty cells was reported as an explanation for the observed misbehavior); one of the five triple faults has been identified as single fault and three as double faults. For the comparison-based model, 18 double faults and two triple faults have been identified as single faults and another two triple faults have been identified as double faults. For the dependency-based model, 19 double faults and two triple faults have been identified as single faults and two triple faults have been identified as double faults. The more abstract a model is, the more likely higher order faults will be reported in single fault diagnoses.

Because the value-based model represents the semantics of a spreadsheet in a direct fashion, it is the most accurate model in terms of computing all diagnoses that really can explain the given misbehavior. Hence, in the evaluation the value-based model can be seen as reference model. The other two models have a good performance compared to the value-based model. There is no clear winner when comparing the dependency-based model with the comparison-based model. The required diagnosis computation time is low in both cases allowing to use both models under real-time conditions. In our comparison there is a slight advantage for the comparison-based model w.r.t. diagnosis time. When comparing the diagnosis output of both models there is only one case where the comparison-based model behaves more similar to the value-based model, i.e., arithmetics02_2_1. The

[1]online available http://spreadsheets.ist.tugraz.at/index.php/corpora-for-benchmarking/integer-corpus/

	Solving time [ms]			**Diagnoses***			**Cardinality**			**Observations**		**Faulty**
	V	**C**	**D**	**V**	**C**	**D**	**V**	**C**	**D**	**Incorrect**	**Correct**	**cells**
arithmetics00_1_1	141	0	1	7	8	8	1	1	1	1	0	1
arithmetics00_1_2	145	1	0	3	5	5	1	1	1	1	1	1
arithmetics00_1_3	138	0	3	5	5	5	1	1	1	1	1	1
arithmetics00_2_1	245	0	1	15	1	1	2	1	1	2	0	2
arithmetics00_2_2	195	1	1	8	8	8	1	1	1	1	0	2
arithmetics00_2_3	144	1	3	4	5	5	1	1	1	1	1	2
arithmetics01_1_1	38	1	4	1	4	4	1	1	1	1	1	1
arithmetics01_1_2	91	0	4	4	4	4	1	1	1	1	1	1
arithmetics01_1_3	585	1	1	11	11	11	1	1	1	1	0	1
arithmetics01_2_1	326	1	1	10	11	11	1	1	1	1	0	2
arithmetics01_2_2	1150	1	6	36	1	3	2	1	1	2	0	2
arithmetics01_2_3	115	0	1	4	4	4	1	1	1	1	1	2
arithmetics01_3_1	122	1	6	6	3	3	2	1	1	2	0	3
arithmetics02_1_1	18944	1	0	13	16	16	1	1	1	1	0	1
arithmetics02_1_2	379	1	1	7	8	8	1	1	1	1	1	1
arithmetics02_1_3	500	1	6	7	7	7	1	1	1	1	1	1
arithmetics02_2_1	5319	0	1	81	90	7	2	2	1	2	0	2
arithmetics02_2_2	462	3	4	5	7	7	1	1	1	1	1	2
cake_1_1	147048	9	34	44	65	65	1	1	1	1	1	1
cake_2_1	143435	7	38	43	64	64	1	1	1	1	2	2
cake_2_2	139707	10	38	20	65	65	1	1	1	1	1	2
cake_2_3	140937	7	43	17	65	65	1	1	1	1	1	2
cake_3_1	737542	29	87	96	186	186	2	2	2	2	2	3
fibonacci_1_1	1843	1	19	2	2	2	1	1	1	1	1	1
fibonacci_2_3	629	4	13	5	8	8	1	1	1	1	1	2
oscars2012_1_4	826	1	3	10	10	10	1	1	1	1	2	1
oscars2012_1_5	504	6	6	1	11	11	1	1	1	1	1	1
oscars2012_2_2	517	1	4	1	11	11	1	1	1	1	1	2
prom_calculator_1_1	341	1	3	13	13	13	1	1	1	1	1	1
prom_calculator_2_1	422	1	3	13	13	13	1	1	1	1	1	2
prom_calculator_2_2	340	0	4	13	13	13	1	1	1	1	1	2
prom_calculator_2_3	349	3	1	13	13	13	1	1	1	1	1	2
prom_calculator_3_1	340	0	0	10	13	13	1	1	1	1	1	3
shares_1_1	963	3	9	1	1	1	1	1	1	1	11	1
shares_1_2	1263	1	10	1	1	1	1	1	1	1	11	1
shares_1_3	1323	3	9	1	1	1	1	1	1	1	11	1
shares_1_4	1588	1	10	1	1	1	1	1	1	1	11	1
shares_1_5	1256	1	7	1	1	1	1	1	1	1	11	1
shares_2_1	41598	6	24	16	17	17	2	2	2	2	10	2
shares_2_2	5681	6	18	2	2	2	2	2	2	8	4	2
shares_2_3	41888	0	24	17	17	17	2	2	2	2	10	2
shares_3_1	49918	15	45	16	17	17	3	3	3	3	9	3
shopping_b1_1_1	521	1	9	15	15	15	1	1	1	1	2	1
shopping_b1_1_2	510	6	9	15	15	15	1	1	1	1	2	1
shopping_b1_2_1	529	3	7	15	15	15	1	1	1	1	2	2
shopping_b1_2_2	513	3	6	15	15	15	1	1	1	1	2	2
shopping_b1_2_3	510	1	7	16	16	16	1	1	1	1	2	2
shopping_b1_3_1	4624	19	24	240	240	240	2	2	2	2	1	3
Average	**31177.2**	**3.4**	**11.6**									

Table 1: Results for Integer Corpus with V representing the value-based model, C representing the comparison-based model and D representing the improved dependency-based model. The cardinality columns indicate the smallest size of found diagnoses.

reason might be that in most cases there is only one output cell with a wrong value. In this case, there seems to be no real advantage about knowing the deviation of values instead of only knowing that an output cell's value is wrong. In order to clarify the initial question whether the comparison-based model is better than the dependency-based models w.r.t. diagnosis accuracy further studies have to be carried out.

Related Work

Jannach and Engler (2010) used constraint solving for spreadsheet debugging. Jannach and Schmitz (2014) extended this work by developing an add-on for Excel, the EXQUISITE debugging tool. Abreu *et al.* (2015) developed a model-based debugging approach for spreadsheets which converts a spreadsheet into a value-based constraint satisfaction problem (CSP). In contrast to Jannach and Schmitz's approach, this approach relies on a single test case, and directly encodes the reasoning about the correctness of cells into the CSP. Hofer *et al.* (2014; 2017) proposed to use dependency-based models for spreadsheet debugging. They additionally showed how to improve the diagnostic performance of dependency-based models. The work presented in this paper directly builds on these value-based and improved dependency-based models.

Ruthruff *et al.* (2003) proposed to use Tarantula, a spectrum-based fault localization (SFL) technique for debugging spreadsheets. Later on, Hofer *et al.* (2015) empirically evaluated the performance of different similarity coefficients for SFL on spreadsheets. Abraham and Erwig (2007) developed a spreadsheet debugger, named GoalDebug, which computes repair suggestions for faulty spreadsheets.

There exist many more debugging and quality assurance techniques for spreadsheets. We refer the interested reader to Jannach *et al.*'s survey paper (2014) which provides an exhaustive overview on different techniques and methods for detecting, localizing, and repairing spreadsheets.

Conclusion

In this paper, we presented a first evaluation of a qualitative algebra model used for fault localization in spreadsheets. The model is based on deviations between computed values and expected values using the qualitative values $<$, $=$, and $>$. For the evaluation, we used some spreadsheets from a public available corpus and compiled them into three different types of models, i.e., the value-based model, the dependency-based model, and the comparison-based model, which relies on qualitative algebra and deviation models. We performed the empirical evaluation using constraint representations of the spreadsheet models. We carried out the constraint solving using the constraint solver Minion.

The empirical evaluation based on 48 faulty spreadsheets showed that the value-based model is the slowest one, followed by the dependency-based model, and the comparison-based model. Hence, the diagnosis runtime of the comparison-based model was the lowest and was also never exceeding 30 ms, allowing its use for real-time debugging applications in practice. The diagnosis output in terms of diagnoses was similar. The comparison-based model and the dependency-based model provided diagnoses close to the ones coming from the value-based model, which serves as reference model. Hence, a further more detailed study has to be carried out using more spreadsheet examples.

Acknowledgments

The work described in this paper has been been funded by the Austrian Science Fund (FWF) project DEbugging Of Spreadsheet programs (DEOS) under contract number I2144 and the Deutsche Forschungsgemeinschaft (DFG) under contract number JA 2095/4-1.

References

Abraham, R., and Erwig, M. 2007. Goaldebug: A spreadsheet debugger for end users. In *29th International Conference on Software Engineering (ICSE)*, 251–260.

Abreu, R.; Hofer, B.; Perez, A.; and Wotawa, F. 2015. Using constraints to diagnose faulty spreadsheets. *Software Quality Journal* 23(2):297–322.

Ausserlechner, S.; Fruhmann, S.; Wieser, W.; Hofer, B.; Spork, R.; Muhlbacher, C.; and Wotawa, F. 2013. The right choice matters! SMT solving substantially improves model-based debugging of spreadsheets. In *2013 13th International Conference on Quality Software*, 139–148.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

Fisher, M. I., and Rothermel, G. 2005. The EUSES spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms. *ACM SIGSOFT Software Engineering Notes* 30(4):1–5.

Gent, I. P.; Jefferson, C.; and Miguel, I. 2006. Minion: A fast, scalable, constraint solver. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006)*.

Hofer, B., and Wotawa, F. 2014. Why does my spreadsheet compute wrong values? In *Proceedings of the International Symposium on Software Reliability Engineering (ISSRE)*, volume 25, 112 –121.

Hofer, B.; Perez, A.; Abreu, R.; and Wotawa, F. 2015. On the empirical evaluation of similarity coefficients for spreadsheets fault localization. *Autom. Softw. Eng.* 22(1):47–74.

Hofer, B.; Hoefler, A.; and Wotawa, F. 2017. Combining models for improved fault localization in spreadsheets. *IEEE Trans. Reliability* 66(1):38–53.

Jannach, D., and Engler, U. 2010. Toward model-based debugging of spreadsheet programs. In *9th Joint Conference on Knowledge-Based Software Engineering (JCKBSE 2010)*, 252–264.

Jannach, D., and Schmitz, T. 2014. Model-based diagnosis of spreadsheet programs - A constraint-based debugging approach. *Automated Software Engineering* 1–40.

Jannach, D.; Schmitz, T.; Hofer, B.; and Wotawa, F. 2014. Avoiding, finding and fixing spreadsheet errors - A survey of automated approaches for spreadsheet QA. *Journal of Systems and Software* 94:129–150.

Nica, I., and Wotawa, F. 2012. Condiag – computing minimal diagnoses using a constraint solver. In *Proceedings of the International Workshop on Principles of Diagnosis (DX)*, 185–191.

Nica, I.; Pill, I.; Quaritsch, T.; and Wotawa, F. 2013. A route to success – a performance comparison of diagnosis algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Panko, R. R. 2008. Thinking is bad: Implications of human error research for spreadsheet research and practice. In *CoRR*.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32(1):57–95.

Ruthruff, J.; Creswick, E.; Burnett, M.; Cook, C.; Prabhakararao, S.; Fisher, II, M.; and Main, M. 2003. End-user software visualizations for fault localization. In *Proceedings of the 2003 ACM symposium on Software visualization (SoftVis '03)*, 123–132. ACM.

Wotawa, F. 2016. On the use of qualitative deviation models for diagnosis. In *29th International Workshop on Qualitative Reasoning (QR)*.