On the Usage of Dependency-based Models for Spreadsheet Debugging

Birgit Hofer
Graz University of Technology
Inffeldgasse 16b/II
8010 Graz, Austria
bhofer@ist.tugraz.at

Franz Wotawa
Graz University of Technology
Inffeldgasse 16b/II
8010 Graz, Austria
wotawa@ist.tugraz.at

ABSTRACT

Locating faults in spreadsheets can be difficult. Therefore, tools supporting the localization of faults are needed. Model-based software debugging (MBSD) is a promising fault localization technique. This paper presents a novel dependency-based model that can be used in MBSD. This model allows improvements of the diagnostic accuracy while keeping the computation times short. In an empirical evaluation, we show that dependency-based models of spreadsheets whose value-based models are often not solvable in an acceptable amount of time can be solved in less than one second. Furthermore, we show that the amount of diagnoses obtained through dependency-based models is reduced by 15 % on average when using the novel model instead of the original dependency-based model. The short computation time and the improved diagnostic accuracy enable the usage of model-based debugging for spreadsheets in practice.

Categories and Subject Descriptors

H.4.1 [**Information Systems Applications**]: Office Automation—*Spreadsheets*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Debugging aids*

Keywords

Spreadsheets, Debugging, Model-based Fault Localization

1. INTRODUCTION

Even for small spreadsheets, the localization of faults can be time consuming and frustrating. Thus, approaches supporting fault localization in spreadsheets are needed. Some fault localization techniques developed for the software engineering discipline have been adapted to the spreadsheet domain, for example [1, 9, 10, 8]. Model-based software debugging (MBSD) [16, 18, 13] is one of these techniques. So far, researchers have only focused on methods that use value-based models [3, 4, 12, 11]. Value-based models compute a small set of possible explanations (i.e., diagnoses) for an observed misbehavior. This small set of diagnoses is helpful for users when debugging. Unfortunately, value-based models have high computation times and they do not scale: the underlying solving mechanisms have problems when dealing with variables with large domains and real numbers. In an empirical evaluation, Außerlechner et al. [5] showed the limitations of different constraint solvers and SMT (satisfiability modulo theories) solvers when using these models.

To the best of our knowledge, dependency-based models have not been used for localizing faults in spreadsheets. The reason for this may be their inaccuracy. Dependency-based models compute a significantly higher number of diagnoses than value-based models. In this paper, we propose a novel type of dependency-based model which uses equivalence instead of the implication to model the dependency relation between cells. This allows improvements on the diagnostic accuracy while keeping the computation times short.

In order to demonstrate the differences between the value-based, the original dependency-based, and our improved dependency-based model, we make use of a running example. This example is a simplified version of the "homework/budgetone" spreadsheet taken from the EUSES spreadsheet corpus [6]. We manually injected a fault into the spreadsheet in cell `D5`. Figure 1a shows the normal (or value) view of this faulty spreadsheet variant. Figure 1b shows the formula view of the same spreadsheet. The output cells[1] of the spreadsheet are shaded in gray. The faulty cell `D5` is highlighted with a red box. The fault manifests in the value of the output cell `D7`. The expected value for this cell is 78,6 %.

	A	B	C	D
1	**Item**	**1st Qtr**	**2nd Qtr**	**Total**
2	**Units Sold**	1000	1500	2500
3	**ASP/Unit**	$ 20	$ 21	$ 20,6
4	**Sales Revenue**	$ 20.000	$ 31.500	$ 51.500
5	**Expenses**	$ 5.000	$ 6.000	$ 5.000
6	**Operating Income**	$ 15.000	$ 25.500	$ 46.500
7	**Op Income in %-Sales**	75,0 %	81,0 %	90,3 %

(a) Normal view

	A	B	C	D
1	**Item**	**1st Qtr**	**2nd Qtr**	**Total**
2	**Units Sold**	1000	1500	=SUM(B2:C2)
3	**ASP/Unit**	20	21	=D4/D2
4	**Sales Revenue**	=B3*B2	=C3*C2	=SUM(B4:C4)
5	**Expenses**	5000	6000	=SUM(B5:B5)
6	**Operating Income**	=B4-B5	=C4-C5	=D4-D5
7	**Op Income in %-Sales**	=B6/B4	=C6/C4	=D6/D4

(b) Formula view

Figure 1: Running example

When using model based debugging, the faulty cell of this example spreadsheet can be detected independent of the underlying model. There are however differences with respect to runtime and solution size. The value-based model and our novel dependency-based model identify three cells that

[1] An output cell is a cell that is not referenced by any other cell.

could explain the observed misbehavior, while the original dependency-based model identifies six cells as possible explanations. When using the constraint solver MINION [7], both dependency-based models require only one third of the computation time compared with the value-based model.

This work is related to the work of Mayer and Stumptner [15] and Wotawa [17]. Mayer and Stumptner evaluated models for model-based debugging in the software domain. Wotawa discussed the relationship of model-based debugging and program slicing.

In the remainder of this paper, we will explain the different types of models and show the model representations for the above described running example (see Section 2). In Section 3, we will empirically compare the different models with respect to efficiency and effectiveness. The novel dependency-based model reduces the number of computed diagnoses by 15 % compared to original dependency-based model. Furthermore, the empirical evaluation shows that the dependency-based models can be solved in less than one second even for those spreadsheets whose value-based models require more than 20 minutes solving time.

2. MODEL-BASED DEBUGGING

In model-based software debugging (MBSD), the cells of a spreadsheet and the given observations (i.e., the test case[2]) are converted into constraints. As the given test case is a failing test case, this constraint system results in a contradiction. In order to determine which cells could resolve this contradiction, MBSD introduces abnormal variables (AB) for each cell. These abnormal variables represent the "health" state of the cells. If a cell c is not abnormal, the formula of the cell must be correct [18]:

$$\neg AB(c) \rightarrow \text{constraint}(c). \tag{1}$$

This logic expression can be transformed to

$$AB(c) \vee \text{constraint}(c). \tag{2}$$

Having such a constraint system, we are able to use a constraint or SMT solver to determine which abnormal variables have to be set to `true` to eliminate the contradiction. We refer the interested reader to [18, 14] for more information about the principles of MBSD. MBSD can be performed using either dependency-based or value-based models. We discuss these models and our novel dependency-based model in the following subsections.

2.1 Value-based models

When using value-based models, the values of the cells are propagated. A value-based constraint system contains (i) the input cells and their values, (ii) the output cells and their expected values, and (iii) all formulas concatenated with their abnormal variable. The constraint representation handles the formulas as equations instead of assignments. This allows to draw conclusions on the input from the output of a formula. Such a value-based model for spreadsheets is proposed by Abreu et al. [3, 4]. The running example from Figure 1b is converted into the following constraints:

[2]A test case is a tuple (I, O), where I are the values for the input cells and O the expected values for the output cells. A test case is a failing test case if at least one computed output value differs from the expected value.

Input:			*Output:*		
$B2$	==	1000	$D3$	==	20.6
$C2$	==	1500	$B7$	==	0.75
$B3$	==	20	$C7$	==	0.81
...			$D7$	==	0.786

Formula constraints:

$$\begin{aligned}
AB(\text{cell}_{D2}) &\vee D2 == B2 + C2 \\
AB(\text{cell}_{D3}) &\vee D3 == D4/D2 \\
AB(\text{cell}_{B4}) &\vee B4 == B3 \times B2 \\
AB(\text{cell}_{C4}) &\vee C4 == C3 \times C2 \\
AB(\text{cell}_{D4}) &\vee D4 == B4 + C4 \\
AB(\text{cell}_{D5}) &\vee D5 == B5 \\
&\dots \\
AB(\text{cell}_{D7}) &\vee D7 == D6/D4
\end{aligned}$$

Solving this constraint system leads to three possible solutions: Either cell D5, D6 or D7 must contain the fault.

2.2 Original dependency-based models

When using dependency-based models, only the information about whether the computed values are correct is propagated. Therefore, all cell values are represented as Boolean instead of Integer or Real values. All variables representing input cells are initialized with `true`. All variables representing correct output cells are also initialized with `true`. The variables representing erroneous output cells are initialized with `false`. Instead of using the concrete formulas in the constraints, only the correctness relation is modeled. If the formula of cell c is correct and the input values of a formula are correct then cell c must compute a correct value:

$$AB(\text{cell}_c) \vee \bigwedge_{c' \in \rho(c)} c' \rightarrow c \tag{3}$$

where $\rho(c)$ is the set of all cells that are referenced in c. Details about this modeling for software written in an imperative language can be found e.g. in [17]. The dependency-based constraints for our running example are as follows:

Input:			*Output:*		
$B2$	==	true	$D3$	==	true
$C2$	==	true	$B7$	==	true
$B3$	==	true	$C7$	==	true
...			$D7$	==	false

Formula constraints:

$$\begin{aligned}
AB(\text{cell}_{D2}) &\vee (B2 \wedge C2 \rightarrow D2) \\
AB(\text{cell}_{D3}) &\vee (D2 \wedge D4 \rightarrow D3) \\
AB(\text{cell}_{B4}) &\vee (B2 \wedge B3 \rightarrow B4) \\
AB(\text{cell}_{C4}) &\vee (C2 \wedge C3 \rightarrow C4) \\
AB(\text{cell}_{D4}) &\vee (B4 \wedge C4 \rightarrow D4) \\
AB(\text{cell}_{D5}) &\vee (B5 \rightarrow D5) \\
&\dots \\
AB(\text{cell}_{D7}) &\vee (D4 \wedge D6 \rightarrow D7)
\end{aligned}$$

Solving this constraint system leads to six possible solutions: Either cell B4, C4, D4, D5, D6 or D7 must contain

the fault. This dependency-based model computes more diagnoses because of the implication. In the value-based model, the cells B4, C4, and D4 can be excluded from the set of possible diagnoses because B4 and C4 are used to compute D4, and D4 is used to compute D3, which is known to compute the correct value. Unfortunately, this information gets lost when using the implication because the implication allows conclusions only from the input to the output but not vice versa. This problem will be solved with the novel dependency-based model that is explained in the following subsection.

2.3 Novel dependency-based models

In order to eliminate the previously described weakness of dependency-based models, we use bi-implication (equivalence) instead of the implication. The formula constraints for our running example from Figure 1b are as follows:

$$\begin{aligned}
AB(\text{cell}_{D2}) &\vee (B2 \wedge C2 \leftrightarrow D2)\\
AB(\text{cell}_{D3}) &\vee (D2 \wedge D4 \leftrightarrow D3)\\
AB(\text{cell}_{B4}) &\vee (B2 \wedge B3 \leftrightarrow B4)\\
AB(\text{cell}_{C4}) &\vee (C2 \wedge C3 \leftrightarrow C4)\\
AB(\text{cell}_{D4}) &\vee (B4 \wedge C4 \leftrightarrow D4)\\
AB(\text{cell}_{D5}) &\vee (B5 \leftrightarrow D5)\\
&\dots\\
AB(\text{cell}_{D7}) &\vee (D4 \wedge D6 \leftrightarrow D7)
\end{aligned}$$

Solving this constraint system leads to the same 3 diagnoses as when using a value-based model.

The bi-implication cannot be used in case of coincidental correctness. Coincidental correctness might occur for example in the following situations:

- usage of conditional function (e.g., the IF-function),
- abstraction function like MIN, MAX, COUNT,
- usage of Boolean,
- multiplication with a 0-value, and
- power with 0 or 1 as base number or 0 as exponent.

Please note, that this list gives only examples. It is not a complete list, because the size of the list depends on the functions that are supported by the concrete spreadsheet environment (e.g. Microsoft Excel, iWorks'Number, OpenOffice's Calc). All formulas where coincidental correctness might happen still have to be modeled with the implication instead of the bi-implication.

3. EMPIRICAL EVALUATION

This section consists of two major parts: the empirical setup (discussing the prototype implementation, the used platform, and the evaluated spreadsheet corpora) and the results showing that dependency-based models are able to compute diagnoses within a fraction of a second even for spreadsheets whose value-based models require more than 20 minutes of solving time. In addition, this empirical evaluation shows that the number of diagnoses obtained by the novel dependency-based model is reduced by 15 % on average compared to the original dependency-based model.

We developed a prototype in Java for performing the empirical evaluation. This prototype uses MINION [7] as a constraint solver. MINION is an out-of-the-box, open source constraint solver and offers support for almost all arithmetic, relational, and logic operators such as multiplication, division, and equality over Boolean and Integers.

The evaluation was performed on an Intel Core2 Duo processor (2.67 GHz) with 4 GB RAM and Windows 7 as operating system. We used the MINION version 0.15. The computation time is the average time over 10 runs.

We evaluated the models by means of spreadsheets from the publicly available Integer spreadsheet corpus[3] [5]. This corpus contains 33 different spreadsheets (12 artificially created spreadsheets and 21 real-life spreadsheets), e.g., a spreadsheet that calculates the lowest price combination on a shopping list or the winner of Wimbleton 2012. These spreadsheets contain both arithmetical and logical operators as well as the functions SUM and IF. The spreadsheets contain on average 39 formula cells, the largest spreadsheet contains 233 formulas. Faulty versions of the spreadsheets (containing single, double and triple faults) were created by randomly selecting formulas and applying mutation operators [2] on them. The corpus contains in total 220 mutants. In the empirical evaluation, we used only the spreadsheets which contain a single fault, i.e. 94 spreadsheets.

Table 1 compares the three types of models with respect to fault localization capabilities and runtimes. The fault localization capabilities are expressed by means of the number of cells that are single fault diagnoses. The runtime is measured by means of MINION's average solving time in milliseconds. The spreadsheets are divided into two subgroups: spreadsheets whose value-based models are solved by MINION in less then 20 minutes and spreadsheets whose value-based models could not be solved within 20 minutes (i.e. 31 from 94 spreadsheets). For these 31 spreadsheets, the dependency-based models are solved in less than one second. These runtime results indicate that dependency-based models are better suited for debugging large spreadsheets than value-based models.

Table 1: Evaluation results

Model	Single fault diagnoses	Solving time (in ms)
63 spreadsheets		
Value-based	4.0	56818.8
Original dep.-based	13.2	32.0
Novel dep.-based	11.0	31.6
31 spreadsheets		
Value-based	-	$>$ 20 minutes
Original dep.-based	45.0	187.4
Novel dep.-based	38.6	164.8

Considering the diagnostic accuracy, the value-based model yields better results. It computes only one third of the diagnoses of the original dependency-based model. The improved dependency-based model computes on average 15 % less diagnoses than the original dependency-based model.

Table 2 gives an overview of the reduction that can be achieved when using the novel instead of the original dependency-based model. The reduction is expressed by means of the following metric:

$$\text{REDUCTION} = 1 - \frac{|\text{Diagnoses in the novel model}|}{|\text{Diagnoses in the original model}|}. \quad (4)$$

[3] https://dl.dropbox.com/u/38372651/Spreadsheets/Integer_Spreadsheets.zip

Table 2: Summary of the achieved reduction when using the novel model instead of the original dependency-based model

Reduction	Number of spreadsheets
0 %	64
]0 %;10 %]	0
]10 %;20 %]	1
]20 %;30 %]	1
]30 %;40 %]	2
]40 %;50 %]	5
]50 %;60 %]	0
]60 %;70 %]	4
]70 %;80 %]	2
]80 %;90 %]	7
]90 %;100 %]	8

For 64 spreadsheets, no reduction in the number of diagnoses was achieved when using the novel dependency-based model instead of the original model. However, for 15 spreadsheets, a reduction of more than 80 % was achieved.

4. DISCUSSION AND CONCLUSIONS

Locating faulty formulas in spreadsheets can be time consuming. This paper addresses the fault localization problem by means of model-based diagnosis. Our most important contribution is the introduction of a novel dependency-based model. This novel dependency-based model improves previous work in two ways: (1) Compared to the original dependency-based model, it reduces the amount of diagnoses that have to be manually investigated by 15 %. (2) Compared to the value-based model, it reduces the required solving time and allows the computation of diagnoses in real-time where the value-based model cannot compute solutions within 20 minutes. The savings in computation time can be explained by the reduction of the domain: The dependency-based model requires only Boolean variables instead of Integers and Real numbers.

The reduction of the domain comes with additional advantages: (1) An arbitrary solver can be used, because all solvers support at least Boolean variables. Even spreadsheets containing Real numbers can be debugged with any solver when using dependency-based models. (2) The user does not need to indicate concrete values for the erroneous output variables. The information that an output cell computes the wrong value is sufficient.

In the description of the model, we listed all types of coincidental correctness occurring in the spreadsheets used in the empirical evaluation. This list is not exhaustive. For using this model in practice, the list has to be extended.

We are convinced that the model presented improves the state of the art in model-based diagnosis. Further work includes a user study and the adaptation to other types of programs, e.g. programs written in imperative or object-oriented languages.

Acknowledgments

The research herein is partially conducted within the competence network Softnet Austria II (www.soft-net.at, COMET K-Projekt) and funded by the Austrian Federal Ministry of Economy, Family and Youth (bmwfj), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT).

5. REFERENCES

[1] R. Abraham and M. Erwig. GoalDebug: A Spreadsheet Debugger for End Users. In *Proceedings of the 29th International Conference on Software Engineering (ICSE 2007)*, pages 251–260, 2007.

[2] R. Abraham and M. Erwig. Mutation Operators for Spreadsheets. *IEEE Transactions on Software Engineering*, 35(1):94–108, 2009.

[3] R. Abreu, A. Riboira, and F. Wotawa. Constraint-based debugging of spreadsheets. In *CibSE'12*, pages 1–14, 2012.

[4] R. Abreu, A. Riboira, and F. Wotawa. Debugging of spreadsheets: A CSP-based approach. In *3th IEEE Int. Workshop on Program Debugging*, 2012.

[5] S. Außerlechner, S. Fruhmann, W. Wieser, B. Hofer, R. Spörk, C. Mühlbacher, and F. Wotawa. The right choice matters! SMT solving substantially improves model-based debugging of spreadsheets. In *Proceedings of the 13th International Conference on Quality Software (QSIC'13)*, pages 139–148. IEEE, 2013.

[6] M. Fisher and G. Rothermel. The EUSES Spreadsheet Corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms. *SIGSOFT Software Engineering Notes*, 30(4):1–5, 2005.

[7] I. P. Gent, C. Jefferson, and I. Miguel. Minion: A fast, scalable, constraint solver. In *Proceedings of the 7th European Conference on Artificial Intelligence (ECAI 2006)*, pages 98–102, 2006.

[8] B. Hofer, A. Perez, R. Abreu, and F. Wotawa. On the empirical evaluation of similarity coefficients for spreadsheets fault localization. *Automated Software Engineering*, pages 1–28, 2014.

[9] B. Hofer, A. Riboira, F. Wotawa, R. Abreu, and E. Getzner. On the Empirical Evaluation of Fault Localization Techniques for Spreadsheets. In *Proceedings of the 16th International Conference on Fundamental Approaches to Software Engineering (FASE 2013)*, pages 68–82, Rome, Italy, 2013.

[10] D. Jannach, A. Baharloo, and D. Williamson. Toward an integrated framework for declarative and interactive spreadsheet debugging. In *Proceedings of the 8th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2013)*, pages 117–124, 2013.

[11] D. Jannach and U. Engler. Toward model-based debugging of spreadsheet programs. In *JCKBSE 2010*, pages 252–264, Kaunas, Lithuania, 2010.

[12] D. Jannach and T. Schmitz. Model-based diagnosis of spreadsheet programs: a constraint-based debugging approach. *Automated Software Engineering*, pages 1–40, 2014.

[13] C. Mateis, M. Stumptner, D. Wieland, and F. Wotawa. Model-based debugging of Java programs. In *Proceedings of AADEBUG*, 2000.

[14] W. Mayer and M. Stumptner. Model-based debugging – state of the art and future challenges. *Electronic Notes in Theoretical Computer Science*, 174(4):61–82, May 2007.

[15] W. Mayer and M. Stumptner. Evaluating models for model-based debugging. In *Proceedings of the International Conference on Automated Software Engineering (ASE)*, 2008.

[16] R. Reiter. A Theory of Diagnosis from First Principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[17] F. Wotawa. On the Relationship between Model-Based Debugging and Program Slicing. *Artificial Intelligence*, 135:125–143, February 2002.

[18] F. Wotawa, M. Nica, and I.-D. Moraru. Automated debugging based on a constraint model of the program and a test case. *The journal of logic and algebraic programming*, 81(4), 2012.