*ECAI 2014*
*T. Schaub et al. (Eds.)*


*doi:10.3233/978-1-61499-419-0-1029*

# Comparing Models for Spreadsheet Fault Localization

**Birgit Hofer** and **Franz Wotawa**[1]

**Abstract.** Locating faults in spreadsheets can be difficult. Therefore, tools supporting the localization of faults are needed. This paper presents a novel dependency-based model that can be used in Model-based software debugging (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, the amount of diagnoses is reduced by 15 % on average when using the novel instead of the original dependency-based model.

## 1 Introduction

Localizing faults in programs is often difficult and time consuming. Model-based software debugging (MBSD) is an automated debugging technique which helps to faster localize faults in software. MBSD originates from model-based diagnosis [5] and is flexible because different models can be used. The chosen model influences the quality of the obtained diagnosis results. The closer the model to the original program behavior, the better the results. Unfortunately, precise models (e.g. value-based models) are computationally demanding. Abstract models (e.g. dependency-based models) have a low computational complexity but they compute more diagnoses.

In this paper, we rely on spreadsheets, but the presented ideas can be easily adapted to functional and procedural languages. Localizing faulty cells can be demanding because spreadsheets lack support for abstraction, encapsulation, or structured programming. Therefore, approaches supporting fault localization in spreadsheets are needed. Recent work [1, 4] introduced MBSD for spreadsheets using value-based models. These approaches do not sufficiently support spreadsheets containing real numbers because of the limitations of the underlying constraint and SMT solvers [2]. To the best of our knowledge, dependency-based models [7] have not been used for localizing faults in spreadsheets, maybe because of their inaccuracy. In this paper, we show how to improve the diagnostic accuracy of dependency-based models while keeping the computation time short.

We make use of a running example to demonstrate the differences between the models. Figure 1(a) shows the normal view of a faulty spreadsheet, Figure 1(b) the formula view. Input cells are shaded in light gray, output cells in dark gray. The faulty cell `D5` is framed. The fault manifests in the output cell `D6`, whose expected value is 12 900. When using any of the discussed models, the faulty cell can be detected. The value-based model and our novel dependency-based model identify three cells that could explain the erroneous output, the original dependency-based model identifies six cells. When using the constraint solver MINION [3], the dependency-based models require only one third of the computation time of the value-based model.

[1] Graz University of Technology, Austria, {bhofer, wotawa}@ist.tugraz.at

| | A | B | C | D |
|---|---|---|---|---|
| 1 | Item | P1 | P2 | Total |
| 2 | Units | 100 | 300 | 400 |
| 3 | Price | 20 | 40 | 35 |
| 4 | Income | 2 000 | 12 000 | 14 000 |
| 5 | Costs | 500 | 600 | 500 |
| 6 | Profit | 1 500 | 11 400 | 13 500 |

(a) Normal view

| B | C | D |
|---|---|---|
| P1 | P2 | Total |
| 100 | 300 | =SUM(B2:C2) |
| 20 | 40 | =D4/D2 |
| =B3*B2 | =C3*C2 | =SUM(B4:C4) |
| 500 | 600 | =SUM(B5:B5) |
| =B4-B5 | =C4-C5 | =D4-D5 |

(b) Formula view

**Figure 1.** Running example

## 2 Model-based Software Debugging

In model-based software debugging (MBSD), the spreadsheet's cells 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. To determine which cells could resolve this contradiction, we use abnormal variables (AB) representing the "health" state of the cells. If cell $c$ is not abnormal, the formula of $c$ must be correct: $\neg\text{AB}(c) \rightarrow \text{constraint}(c)$ (or $\text{AB}(c) \vee \text{constraint}(c)$). 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. In the following, we discuss the different types of models. All models can be automatically obtained from the spreadsheets without human interaction. Since the models are derived from the faulty spreadsheet, they also contain the fault(s).

**Value-based model.** 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 allows us 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. [1]. The running example from Figure 1(b) is converted into the following constraints:

| *Input:* | *Output:* | *Formula constraints:* |
|---|---|---|
| $B2 == 100$ | $D3 == 35$ | $AB(\text{cell}_{D2}) \vee D2 == B2 + C2$ |
| $C2 == 300$ | $B6 == 1500$ | $AB(\text{cell}_{D3}) \vee D3 == D4/D2$ |
| $B3 == 20$ | $C6 == 12000$ | $AB(\text{cell}_{B4}) \vee B4 == B3 \times B2$ |
| ... | $D6 == 12900$ | ... |

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

**Original dependency-based model.** When using dependency-based models, only the information about whether the computed values are correct is propagated. All variables representing input cells and correct output cells are initialized with `true`. The variables representing erroneous output cells are initialized with `false`. Instead

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

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: $AB(\text{cell}_c) \vee \bigwedge_{c' \in \rho(c)} c' \rightarrow c$ where $\rho(c)$ is the set of all cells that are referenced in $c$. Details about this modeling technique can be found in [7]. The dependency-based constraints for our running example are as follows:

| *Input:* | *Output:* | *Formula constraints:* |
|---|---|---|
| $B2 == \text{true}$ | $D3 == \text{true}$ | $AB(\text{cell}_{D2}) \vee (B2 \wedge C2 \rightarrow D2)$ |
| $C2 == \text{true}$ | $B6 == \text{true}$ | $AB(\text{cell}_{D3}) \vee (D2 \wedge D4 \rightarrow D3)$ |
| $B3 == \text{true}$ | $C6 == \text{true}$ | $AB(\text{cell}_{B4}) \vee (B2 \wedge B3 \rightarrow B4)$ |
| ... | $D6 == \text{false}$ | ... |

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 result. 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.

**Novel dependency-based model.** The novel dependency-based model uses bi-implication instead of the implication in order to eliminate the previously described weakness. The rationale here is that if a cell value is correct also the contributing parts have to be correct. For example, the constraint for cell D2 from our running example is $AB(\text{cell}_{D2}) \vee (B2 \wedge C2 \leftrightarrow D2)$. Solving the constraint system of the running example leads to the same three diagnoses as when using a value-based model. The bi-implication cannot be used in case of coincidental correctness [6]. In case of coincidental correctness, an output value could be correct even when not all of its input values are correct (fault masking). Coincidental correctness might occur for example when using conditional functions (e.g., IF), abstraction functions (e.g., MIN, MAX, COUNT), Booleans, multiplications with zero, or power with zero or one as base number or zero as exponent. Depending on the concrete functions supported by the used spreadsheet environment (e.g. Microsoft Excel, iWorks'Number, OpenOffice's Calc) this list has to be extended. All formulas where coincidental correctness might happen still have to be modeled with the implication instead of the bi-implication.

## 3 Empirical Evaluation

We developed a prototype in Java which uses MINION [3] as a constraint solver. We evaluated the models using the 94 single fault spreadsheets from the publicly available Integer spreadsheet corpus [2]. The spreadsheets are divided into two sub-groups: spreadsheets whose value-based models are solved by MINION in less than 20 minutes and spreadsheets whose value-based models could not be solved within 20 minutes (i.e. 31 from 94 spreadsheets). Table 1 compares the three types of models with respect to fault localization capabilities and runtimes for these spreadsheets. The fault localization capabilities are expressed by means of the number of cells that are single fault diagnoses. Considering the diagnostic accuracy, the value-based model yields better results. The improved dependency-based model decreases the number of computed diagnoses by 15 % compared to the original dependency-based model. The reduction indicates the average percentage of formula cells that are contained in the set of diagnoses. The runtime is measured by means of MINION's average solving time over 10 runs. For the 31 spreadsheets whose value-based models could not be solved within 20 minutes, the dependency-based models are solved in less than one second.

**Table 1.** Evaluation results

| Model | Single fault diagnoses | Reduction in % | Solving time (in ms) |
|---|---|---|---|
| **63 spreadsheets** | | | |
| Value-based | 4.0 | 53.5 | 56818.8 |
| Original dep.-based | 13.2 | 40.0 | 32.0 |
| Novel dep.-based | 11.0 | 46.0 | 31.6 |
| **31 spreadsheets** | | | |
| Value-based | - | - | $>$ 20 minutes |
| Original dep.-based | 45.0 | 40.0 | 187.4 |
| Novel dep.-based | 38.6 | 58.8 | 164.8 |

## 4 Discussion and Conclusions

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 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. (2) Spreadsheets containing Real numbers can be debugged. (3) 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.

## 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).

## REFERENCES

[1] Rui Abreu, Andre Riboira, and Franz Wotawa, 'Constraint-based debugging of spreadsheets', in *CibSE'12*, pp. 1–14, (2012).
[2] Simon Außerlechner, Sandra Fruhmann, Wolfgang Wieser, Birgit Hofer, Raphael Spörk, Clemens Mühlbacher, and Franz Wotawa, 'The right choice matters! SMT solving substantially improves model-based debugging of spreadsheets', in *QSIC'13*, pp. 139–148. IEEE, (2013).
[3] Ian P. Gent, Chris Jefferson, and Ian Miguel, 'Minion: A fast, scalable, constraint solver', in *ECAI 2006*, pp. 98–102, (2006).
[4] Dietmar Jannach and Thomas Schmitz, 'Model-based diagnosis of spreadsheet programs: a constraint-based debugging approach', *Automated Software Engineering*, 1–40, (2014).
[5] Raymond Reiter, 'A Theory of Diagnosis from First Principles', *Artificial Intelligence*, **32**(1), 57–95, (1987).
[6] Xinming Wang, Shing-Chi Cheung, Wing Kwong Chan, and Zhenyu Zhang, 'Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization', in *ICSE*, pp. 45–55, (2009).
[7] Franz Wotawa, 'On the Relationship between Model-Based Debugging and Program Slicing', *Artif. Intelligence*, **135**, 125–143, (Feb. 2002).