On the Effectiveness of Automatically Inferred Invariants in Detecting Regression Faults in Spreadsheets

Roy, Sohon; van Deursen, Arie; Hermans, Felienne

DOI
10.1109/QRS-C.2018.00046

Publication date
2018

Document Version
Accepted author manuscript

Published in
Companion of the 18th IEEE International Conference on Software Quality, Reliability, and Security

Delft University of Technology
Software Engineering Research Group
Technical Report Series

On the Effectiveness of Automatically Inferred Invariants in Detecting Regression Faults in Spreadsheets

Sohon Roy, Arie van Deursen, Felienne Hermans

Report TUD-SERG-2018-002

TUD-SERG-2018-002

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Van Mourik Broekmanweg 6
2628 XE Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Note: Accepted for publication in the Companion of the 18th IEEE International Conference on Software Quality, Reliability, and Security.

On the Effectiveness of Automatically Inferred Invariants in Detecting Regression Faults in Spreadsheets

Sohon Roy, Arie van Deursen, Felienne Hermans
Dept. of Software and Computer Technology
Delft University of Technology
Delft, The Netherlands
{S.Roy-1, Arie.vanDeursen, F.F.J.Hermans}@tudelft.nl

***Abstract*—Automatically inferred invariants have been found to be successful in detecting regression faults in traditional software, but their application has not been explored in the context of spreadsheets. In this paper, we investigate the effectiveness of automatically inferred invariants in detecting regression faults in spreadsheets. We conduct an exploratory empirical study on eight spreadsheets taken from VEnron and EUSES corpora. We apply automatic invariant inference to them, create tests based on the inferred invariants, and finally seed the sheets with faults. Results indicate that the effectiveness of the inferred invariants, in terms of accuracy of fault detection, largely varies from spreadsheet to spreadsheet. The effectiveness is found to be affected by the formulas and data contained in the spreadsheets, and also by the type of faults to be detected.**

***Keywords*—Spreadsheets, Invariant Analysis, Regression Faults, Fault Detection, Software Quality, End-user Development**

I. Introduction

Invariant-based Testing is a spreadsheet testing approach [1], where values of spreadsheet cells are tested (checked) by means of conditional formulas. Based on adherence or violation of invariant properties of a spreadsheet, the conditional formulas evaluate to either true or false, indicating passing or failing of the tests. The approach is practiced in its manual form, where the invariants are manually specified, and the conditional test-formulas are manually created. However, manually specifying invariants, and creating tests based on them is time-consuming for the users [1]. It is a motivation for developing automatic approaches in which automatically inferred invariants would play a central role. Such approaches could benefit from already existing invariant inference techniques such as Daikon [2].

Since automatic invariant inference techniques analyze the existing state of a program to infer invariants, tests based on such invariants are expected to be particularly suitable for detecting *regression faults*—faults that cause software to stop functioning as intended after it has been changed. Incidentally, automatically inferred invariants have been found useful for detecting regression faults in conventional software systems [3]. However, the idea of using them has not been explored in the context of spreadsheets. Therefore, in this paper, our goal is to investigate the effectiveness of automatically inferred invariants for the purpose of detecting regression faults in spreadsheets. Also, in this paper, we use the term *inferred invariant* to imply *automatically* inferred invariant, unless specified otherwise.

To fulfill our goal, we conduct an exploratory empirical study with eight spreadsheets from available spreadsheet corpora VEnron [4] and EUSES [5]. We use the invariant inference tool Daikon [2] to automatically infer invariants from the spreadsheets. Next, we augment the spreadsheets with tests based on the inferred invariants. Finally, to mimic regression faults, we seed the augmented spreadsheets with faults according to a fault model based on a set of spreadsheet mutation operators [6]. To mitigate biasing in the fault seeding process, instead of seeding manually, we develop a spreadsheet mutation tool called *Sprutagen*. We then observe the effectiveness of the invariant-based tests in detecting those faults. As such, we measure the accuracy of fault detection, and investigate the factors that may be affecting the accuracy. Results show:

- The accuracy of inferred invariants in detection of regression faults, based on the recall rate of fault detection, shows extreme variation (0-88%) from spreadsheet to spreadsheet, with mean recall of 37%.
- The accuracy is affected by the type of formulas and data prevalent in the spreadsheets; which in turn are responsible for the inference of specific types of invariants. The accuracy is also affected by the type of faults to be detected.

These results imply that inferred invariants are selectively effective; they could be employed on spreadsheets of particular varieties, containing specific types of formulas and data, which allow the inference of more effective type of invariants. Detailed discussion of the types are presented in subsequent sections of this paper. As such, our findings allow for an initial assessment of the potential of inferred invariants. The contributions of this paper are:

1) Findings of an exploratory study, assessing the effective-

	A	B	C	D	F	G	H	I	J	K
7	Prod.	Contract	Contract	Fixed	Fuel	Settled	Futures	(Gains) and Losses		
8	Date	Number	Name	Price	Dth	Price	Price	Total	Realized	Unrealized
9						El Paso SJ		(4)	(4)	(4)
10	Jun-98		Avista	$ 2.22	7,500	$ 1.82		$ 3,000.00	$ 3,000.00	
11	Jul-98		Avista	$ 2.22	7,750	$ 1.86		$ 2,790.00	$ 2,790.00	
12	Aug-98		Avista	$ 2.22	7,750	$ 1.81		$ 3,177.50	$ 3,177.50	
13	Sep-98		Avista	$ 2.22	7,500	$ 1.55		$ 5,025.00	$ 5,025.00	
14	Oct-98		Avista	$ 2.22	7,750	$ 1.67		$ 4,262.50	$ 4,262.50	
15	Nov-98		Avista	$ 2.22	7,500	$ 1.88		$ 2,550.00	$ 2,550.00	
16	Dec-98		Avista	$ 2.22	7,750	$ 1.96		$ 2,015.00	$ 2,015.00	
17	Jan-99		Avista	$ 2.22	7,750	$ 1.72		$ 3,875.00	$ 3,875.00	
18	Feb-99		Avista	$ 2.22	7,000	$ 1.63		$ 4,130.00	$ 4,130.00	
19	Mar-99		Avista	$ 2.22	7,750	$ 1.51		$ 5,502.50	$ 5,502.50	
20	Apr-99		Avista	$ 2.22	7,500	$ 1.59		$ 4,725.00	$ 4,725.00	
21	May-99		Avista	$ 2.22	7,750	$ 2.03		$ 1,472.50	$ 1,472.50	
22					91,250			$ 42,525.00	$ 42,525.00	$ -
23										

Fig. 1. A portion of a typical Excel spreadsheet, with the data targeted for invariant inference highlighted in orange

ness of inferred invariants in detecting regression faults in spreadsheets.

2) A discussion of the factors affecting the effectiveness of inferred invariants, based on the outcomes of the above study.
3) Conception of an approach that would employ inferred invariants, as part of automatizing the existing manual practice of invariant-based spreadsheet testing.

II. Background

A. Spreadsheet Invariants

An invariant is defined as a logical property of a program that holds true during the entirety or some part of its execution [7]. In the context of spreadsheets, invariants are properties of the cells of a spreadsheet that hold true throughout the context of the spreadsheet's usage [8]. The properties are typically expressed in terms of the *labels* of the applicable cells, either individually, or on the basis of rows or columns. For example, in case of a financial spreadsheet, *"**TotalCredit** must be equal to **TotalDebit**"* is an invariant which will be applicable to cells labelled as ***TotalCredit*** and ***TotalDebit***. In this case, the invariant is originating from a domain-specific rule that is established in the domain of finance.

As a second example, consider the spreadsheet partly shown in Figure 1. For this spreadsheet, *"**Settled Price** is always greater than **1.51**"* is an invariant. It is evident that this invariant represents the inherent property of the data that is present in the spreadsheet. Other examples of invariants include properties of column data such as *Upper Bound, Lower Bound, Range, set of permitted values referred to as "One of"* and also interrelations between column data. As such, spreadsheet invariants can originate based on 1) domain-specific, or context-specific rules, specified by the users, as well as on 2) the inherent properties of the data inside spreadsheets.

B. Spreadsheet Mutation Operators

Seeding faults based on a fault model is a commonly applied strategy for evaluation of testing or fault detection approaches [3], [9]. Hence in our study, we seed faults based on a fault model that uses existing spreadsheet *mutation* operators developed by Abraham and Erwig [6] as its basis.

In *mutation testing* [10], faults are seeded into programs to be tested, generating slightly different programs called *mutants*. Test cases need to be created such that the original programs yield test outcomes that are different from those yielded by the mutants, which is referred to as *killing* of the mutants. Taking into account the *competent programmer hypothesis* and *coupling effect* assumptions [11], the seeded faults are assumed to represent faults introduced by programmers, and hence, a test suite that kills mutants is assessed to be effective in detecting faults. As such, mutation testing allows for an evaluation of the effectiveness of testing and fault detection approaches.

Mutation operators are a set of operators in a programming language, that are able to mutate a program in that language, i.e., introduce faults into the program to generate mutants [12], [13].

Abraham and Erwig [6] developed a set of mutation operators for spreadsheets. Their mutation operators reflect error types reported in the spreadsheet literature [14], [15], [16]. They have also been used for evaluating spreadsheet test suites and spreadsheet debugging tools [6]. Hence, we use them as a basis of our fault model, to mimic regression faults, in our study of effectiveness of inferred invariants. A further description of our fault model is provided in Section IV-C3, as part of our study design.

III. Invariant-based Spreadsheet Testing With Automatically Inferred Invariants

A. Automatically Inferring Spreadsheet Invariants

Automatic inference of program invariants is a popular research topic with several techniques and tools proposed in the past. For the purpose of our study, in this paper, we use the Daikon approach [2].

Similar to other approaches employing inferred invariants for regression testing in conventional software [3], the idea of employing inferred invariants for detecting regression faults in spreadsheets is based on the following two assumptions:

1) **Relative Correctness**: Due to the error-proneness of spreadsheets, it is not realistic to expect a spreadsheet to be in a state of absolute correctness. Yet, for the invariant inference to be successful, it must be assumed that the spreadsheet under consideration has a sufficient level of correctness, at least, within the context of its usage. This can be adjudged by the users concerned. This allows for the inference of invariants that will potentially detect relative deviations of the spreadsheet from its existing state and behavior, such as those represented by regression faults.

TABLE I
EXAMPLE OF INFERRED INVARIANTS

Total == Realized
ContractName one of { “Avista”, “Engage” }
Fixed == 2.22
Settled >= 1.51
Fixed >Settled

Total == Realized	Fixed == 2.2	Settled>=1.51	Fixed > Settled
OK	OK	OK	OK
OK	OK	OK	OK
OK	OK	OK	OK
OK	OK	OK	OK
OK	OK	OK	OK

Fig. 2. Example of a test sheet with invariant-based tests copied down along columns

2) **No Functional Change**: A set of invariants, once inferred, will be valid as long as there are no changes of functionality proactively introduced into the spreadsheets. When there are substantial and deliberate functional changes, like additions or deletions of columns, or intentional changes of the formulas, the corresponding invariants are likely not valid anymore and need to be updated.

Daikon is capable of processing spreadsheet data saved in `.csv` format, but there are practical limitations due to which spreadsheets occurring in real life [17], with structural organization as found in Figure 1, cannot be directly fed into Daikon. For the approach to work, the blocks of data we are interested in, have to be re-structured in an acceptable format. That implies converting data blocks into a single, contiguous, and uninterrupted tabular structure with acceptable column headers. Automatizing this conversion process is dependent upon ongoing research [17], [18], [19], [20], and is not within the scope of this paper.

Daikon infers a large number of invariants. Therefore, it offers filters that are used to suppress its output from reporting of invariants that “*are true, but not considered interesting usually because the invariants are considered obvious or redundant in a given context*.”[1] Accordingly, we tune a number of these filter settings to obtain sets of relevant and meaningful invariants for spreadsheets. Once customized, the same settings are used throughout the entire course of the study, and is available for reproducibility.

Eventually, for a spreadsheet like the example partly shown in Figure 1, we obtain the list of invariants shown in Table I after Daikon has been fed with re-structured data.

B. Invariant-based Testing of Spreadsheets

1) Invariant-based Test Creation: As discussed in a recent study on spreadsheet testing methods in practice [1], invariant-based testing involves conditional formulas intended to validate data, or the outputs of other functionality-related formulas inside spreadsheets. The conditions for positive evaluation in such test formulas are the invariant properties of the spreadsheets. Spreadsheet users though do not use the term “invariant” while speaking about such tests; more commonly used terms are “sanity checks”, “cross checks”, “validation rules”, etc.

In practice, the invariants are manually specified by the users, and they are typically based on domain-specific rules (see Section II-A). Yet, manually specifying invariants is time-consuming and an overhead for the users, often discouraging them from testing at all [1]. Our focus therefore, is on invariants that are automatically inferred by a tool such as Daikon.

As illustration, consider the case partly shown in Figure 1. Here, there are no easily identifiable domain-specific invariants. Running Daikon on shaded portion of the data (not entirely shown in figure) we obtain a list of inferred invariants as shown in Table I. For example, let us consider the invariant “$Settled >= 1.51$” where “*Settled*” is representing column G with the header “*Settled Price El Paso SJ*”. The corresponding test for the first applicable cell in that column would be `=IF(G10>=1.51, "OK", "ERROR")`. This test is replicated and copied down for all the rest of the applicable cells in that column. Such tests are often created in a different worksheet than the one which is being tested as partly shown in Figure 2. Inspection of this “test-sheet” then helps the users to understand the results of the tests.

2) Invariant-based Test Coverage: The idea of coverage in case of invariant-based tests is related to the concept of formula *precedence* in spreadsheets. Cells, that are referred to in a particular formula, are termed as *direct precedents* of that formula; direct precedents of precedents, in turn, are termed as *indirect precedents* of the original formula. An invariant-based test formula, in effect, tests the outputs of its direct and indirect precedents. The coverage of an invariant-based test therefore depends upon its direct and indirect precedents [21].

In this paper we are interested in the effectiveness of the invariants, and not a comparative evaluation of the techniques of invariant inference. Therefore, test coverage of a whole spreadsheet, which is dependent on the number of tests, and therefore, on the number of invariants inferred, is not within the scope of this paper. As such, in our study, we only consider faults that are potentially detectable by the tests based on Daikon inferred invariants, i.e. faults occurring only in the areas which are under coverage of the tests.

IV. STUDY DESIGN

A. Research Questions

The objective in this paper is to investigate the effectiveness of inferred invariants in detecting regression faults in spreadsheets. As such, we seek answers to the following research questions:

[1] http://plse.cs.washington.edu/daikon/download/doc/daikon.html#Invariant-filters

TABLE II
DETAILS OF SHEETS USED

Source	File group or name	Sheet name	SheetID
VEnron	335_6_fixed	Avista_1	SH01
VEnron	335_6_fixed	Avista_2	SH02
VEnron	148_3_ces	Devon Noms	SH03
VEnron	319_9_pasadena	Invoice	SH04
EUSES	filby/DNA	Calculations	SH05
EUSES	filby/CHOFAS	Calculations	SH06
EUSES	filby/CHARGED	Calculations	SH07
EUSES	filby/HENON	HenonMap	SH08

RQ1: How accurate are inferred invariants in detecting regression faults in spreadsheets?

RQ2: What factors affect the accuracy of inferred invariants in detecting regression faults in spreadsheets?

B. Experimental Setup

Our setup consists of eight worksheets obtained from spreadsheet workbooks of the two publicly available spreadsheet corpora VEnron [4] and EUSES [5], as shown in Table II.

The spreadsheets in the VEnron corpus have been analyzed and grouped into evolution groups. Items in each group are considered as successive versions of one single spreadsheet. Additionally, the authors of the VEnron corpus provide data on error trends. They identified evolution groups where Excel specific spreadsheet errors were observed to have increased across versions.

Recall that for the inference of invariants, we need to assume a sufficient level of initial correctness for the original spreadsheet (Section III-A). Hence the error-prone evolution groups in the VEnron corpus with zero errors in the initial version provide us with 1) sufficient level of initial correctness, and 2) potentially error-prone spreadsheets.

While the VEnron spreadsheets offer us a type of potentially error-prone spreadsheets, we wanted to also include in our study, spreadsheets from the other end of the spectrum i.e. ones which are expected to be less error prone. Towards this end we selected spreadsheets from the "filby" folder of the EUSES [5] corpus of spreadsheets. The EUSES corpus has been widely used in spreadsheet research and the "filby" folder contains examples of spreadsheet modeling from a book [22]. Thus, we expect the spreadsheets to be more idealistic than the industrial ones from VEnron.

Note that Excel based spreadsheet workbooks contain multiple "worksheets". Since the inference of invariants is related to one worksheet at a time, we conduct our study on single worksheets. Also, henceforth in this paper, by the term *sheet* we imply worksheet, unless specified otherwise.

Our sheet selection is governed further by the feasibility of re-structuring the sheets for processing by Daikon as explained in Section III-A. Eventually we obtain eight sheets for our study as shown in Table II.

C. Experiment

Our experimental procedure consists of three steps that are repeated for each sheet, as described in the following subsections.

TABLE III
FAULT SEEDING BASED ON SPREADSHEET MUTATION OPERATORS

Fault Type	SH01	SH02	SH03	SH04	SH05	SH06	SH07	SH08	Total
AOR	2	2	2	2	2	0	0	2	**12**
CRP	2	2	0	1	2	2	2	2	**13**
CRR	2	2	2	2	2	0	2	2	**14**
LCR	0	0	0	0	2	2	0	0	**4**
ROR	0	0	0	0	2	2	2	0	**6**
RCR	2	2	0	1	2	2	2	2	**13**
FDL	2	2	2	2	2	2	2	2	**16**
FRC	2	2	2	2	2	2	2	2	**16**
RFR	2	2	2	2	2	2	2	2	**16**
UOI	2	2	2	2	2	2	2	2	**16**
CRS	0	0	0	0	2	2	2	0	**6**
NRS	0	0	0	0	0	0	0	0	**0**
CRE	0	0	0	0	2	2	2	0	**6**
NRE	0	0	0	0	0	0	0	0	**0**
RRR	0	0	0	0	0	0	0	0	**0**
FFR	2	2	2	0	2	2	2	0	**12**
DIP	2	2	2	2	2	2	2	2	**16**
Total	**20**	**20**	**16**	**16**	**28**	**24**	**24**	**18**	**166**

1) Invariant Inference: In this step we use Daikon to automatically infer invariants from the sheets. For each sheet we follow the approach described in Section III-A. Daikon provides a list of invariants, similar to Table I.

2) Creation of Invariant-based Tests: In this step we filter out invariants that we assess as obviously irrelevant or coincidental, and convert the relevant invariants into tests, following the approach described in Section III-B1. These tests are added in separate worksheets, referred to as "test sheets", and attached to the original sheets, to produce augmented versions. The test sheets resemble Figure 2, and the coverage areas of the tests are noted to be used in the fault seeding step explained in the following subsection. As explained in Section III-B2, we are interested only in the area of spreadsheets covered by the tests.

3) Fault Seeding: In this step we inject faults into the versions of sheets augmented with tests, using a fault model. As discussed previously in Section II-B, the basis of our fault model is a set of spreadsheet mutation operators designed by Abraham and Erwig [6]. The operators when applied to spreadsheet formulas, generate mutant variants of the formulas representing seeded faults.

We exclude the operator ABS or Absolute Value Insertion, as we believe it is not sufficiently representative of a regression fault that can be introduced by a spreadsheet user. We add a new fault type, the Data I/P error or DIP, to reflect errors caused by entry of wrong input data. As a result, our fault model consists of 17 fault types; they are shown in Table III along with names of the mutation operators.

To avoid chances of bias in a manual seeding process, we have implemented the mutation operators into a tool called Sprutagen (*Spr*eadsheet M*utagen*) which is capable of performing the fault seeding automatically. The seeding process is based on the applicability of the mutation operators. Not all operators are applicable to every type of spreadsheet

TABLE IV
PRECISION AND RECALL OF FAULT DETECTION BY TESTS BASED ON INFERRED INVARIANTS

SheetID	#FN	#FP	#TP	Precision (%)	Recall (%)
SH01	11	0	9	100	45
SH02	11	0	9	100	45
SH03	11	0	5	100	31
SH04	2	0	14	100	88
SH05	27	0	1	100	4
SH06	21	0	3	100	13
SH07	24	0	0	100	0
SH08	5	0	13	100	72
Total	**112**	**0**	**54**	**(mean) 100**	**(mean) 37**

formulas, e.g. CRE or Contiguous Range Expansion is only applicable to formulas that have reference to a contiguous range of cells.

Sprutagen seeds two faults for each fault type of our fault model, unless prevented due to lack of applicability. Table III shows the details of the number of faults seeded for each sheet, and each type of fault. If a seeded fault is detected, it is indicated by the corresponding invariant-based test in the respective "test-sheet".

V. ACCURACY OF INFERRED INVARIANTS IN DETECTING REGRESSION FAULTS

To observe the accuracy of inferred invariants in detecting regression faults (RQ1), we record Precision and Recall, defined as follows:

Precision is the fraction of detected faults that are indeed seeded faults:

$$\frac{TP}{TP+FP}$$

Recall is the fraction of seeded faults that is actually detected by the inferred invariants:

$$\frac{TP}{TP+FN}$$

Table IV, shows the number of false negatives, false positives, true positives, as well as the percentages for precision and recall related to fault detection for each sheet.

The precision is 100%, in each case, meaning that all the detected faults, are indeed seeded faults. This follows from the fact that in the context of our study, where the only change we introduced in the sheets are the faults, there is no possibility of a false positive. In a real-world context, however, where users are expected to also introduce legitimate changes into the spreadsheets, a high precision value can be ensured by the customization of Daikon output (Section III-A), and the subsequent manual filtering we do as part of our experiment step (Section IV-C2). This eliminates invariants that are irrelevant in context of the usage of a particular spreadsheet. A similar finding is also discussed by Mirshokraie and Mesbah [3] in their study of using invariants for regression testing in Javascript. Therefore, precision is not the practically significant indicator of the accuracy, and in the rest of the paper, we use recall as the indicative measure of accuracy of inferred invariants.

From Table IV we observe that the mean recall is 37%, and the individual recall for each sheet varies between 0-88%, which deserves attention. In some cases, such as sheet SH04, the inferred invariants detect almost all the faults, whereas in other cases, like sheet SH07, they fail to detect any of the seeded faults. We examine this observation in further detail in the following section where we explore factors affecting the accuracy of inferred invariants.

To summarize, as answers to RQ1, we conclude:

> The practically significant indicator of the accuracy of inferred invariants is the recall rate of fault detection. It shows large variation from spreadsheet to spreadsheet, ranging between 0-88%, with an average of 37%.

VI. FACTORS AFFECTING THE ACCURACY OF INFERRED INVARIANTS IN DETECTING REGRESSION FAULTS

For our second research question (RQ2), we investigate which factors affect the accuracy of inferred invariants. Based on the answers to our first research question (RQ1), as explained in Section V, we use the recall rate of fault detection as the practically significant indicator of accuracy of inferred invariants.

We investigate the factors from two perspectives. First, those related to characteristics of the spreadsheets, and second, those related to the characteristics of the faults. Finally, we summarize our conclusions for RQ2.

A. Spreadsheet Characteristics

The characteristics of the spreadsheets, in particular the types of formulas and data present in them, lead to the inference of different types of invariants. In our study, we were able to observe the following types of invariants:

- Fixed Value - e.g. cells in column *Price* should always be 2.45
- Upper Bound - e.g. cells in column *Price* should always be greater than 2.45
- Lower Bound - e.g. cells in column *Price* should always be less than 2.45
- Range - e.g. cells in column *Price* should always lie between 2.45 and 5.55
- Column Interrelation - e.g. cells in column *Price* should always be greater than cells in column *Cost of Production*
- One of - e.g. cells in column *Allowed States* should always be one of {0,1,2,3}

Table V shows the sheets sorted and grouped into three groups based on their recall rate of fault detection. It shows the source, the domain, and the dominating type of invariant for each sheet. As seen from Table V, the dominating type of invariant found in the sheets appears to be a factor affecting the accuracy of fault detection. We explore this further by investigating the groups that we call A and C, with the highest and lowest recall rates respectively.

TABLE V
THE SHEETS SORTED AND GROUPED ACCORDING TO DECREASING RECALL (%) OF FAULT DETECTION

Recall Based Group	SheetID	Recall (%)	Source	Domain	Dominating Type of Invariants
A (70-90%)	SH04	88	VEnron	Business	Fixed Value (5/5)
	SH08	72	EUSES	Physics	Range with very precise boundaries (2/4) Upper Bound with very precise boundary (2/4)
B (30-50%)	SH01	45	VEnron	Business	Column interrelations (2/4)
	SH02	45	VEnron	Business	Fixed Value (1/3) Column interrelations (1/3) Lower Bound (1/3)
	SH03	31	VEnron	Business	Lower Bound (4/5)
C (0-20%)	SH06	13	EUSES	Biotechnology	Range (4/11)
	SH05	4	EUSES	Genetics	One of (3/5)
	SH07	0	EUSES	Genetics	One of (6/6)

1) Group A (70-90%): : Examining SH04 we find that the columns are replete with duplication formulas. For example, the cells of column M—M3, M4, and M5—contain formulas `=M2`, `=M3`, and `=M4` respectively, and this pattern is continued for the whole column. These formulas are sequentially copying the values of their previous cells in the same column. Effectively, this has resulted in all the cells in a column to have the same values. Consequently, the invariants inferred from this sheet are not Upper Bound, Lower Bound, or Range type invariants, but invariants of Fixed Value like "$Fixed\ Price == 1500.0$". Due to this, even the smallest of deviations from the fixed values cause invariant violations, and the faults are detected, resulting in a high recall rate of 88%.

Likewise in SH08, the formulas are used for highly precise mathematical calculations. The resulting invariants are therefore of the type

$$0.381545407380169 >= y2 >= -0.384254894160137$$

or,

$$x2 <= 1.27181802460056$$

As evident, the ranges bounded by values with such high precision after the decimal point, leave extremely small margins for deviations.

From the overall observation of group A, we see:

> Fixed Value type of invariants, and Range or Upper Bound type of invariants with highly precise boundaries ($\approx$ 10 decimal places), are most effective in terms of recall rate of fault detection, with rates ranging between 72-88%.

2) Group C (0-20%): : In this group, SH06 contains 4 Ranges with standard boundaries ($\approx$ 2-3 decimal places), out of a total of 11 invariants, and the ranges are not with highly precise boundaries like seen in Group A.

SH07 with the lowest recall in this group, contains formulas of the form `IF(A13="K", 0,1)`, and consequently, the invariants inferred are also of the form

$$Histidine\ is\ one\ of\ 0.0, 1.0$$

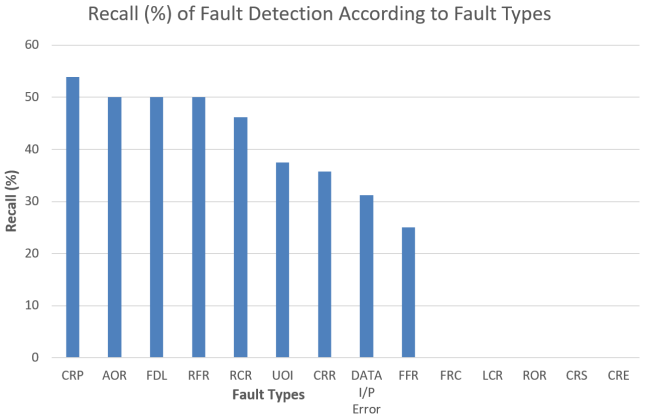

Fig. 3. Variation of Recall (%) of fault detection by inferred invariants, depending on fault types based on spreadsheet mutation operators

As such, most of the faults seeded, effectively switch the values of the cells between 1 and 0. Neither causes a violation of the invariants, and thus the faults remain undetected.

In the case of SH05 also, 3 out of 5 invariants are of the type "One of". Invariants of the "One of" type providing a set of permitted values for a column, show least recall, as faults often switch the value of a cell from one permitted value to another permitted value.

From overall observation of group C, we note:

> "One of" type invariants, specifying a set of permitted values, are least effective in terms of recall rate of fault detection, with rates as low as 0-13%.

Thus, we can observe that the accuracy depends on the type of the invariants, which in turn depend upon the formulas and data in the spreadsheets.

B. Fault Characteristics

In contrast to Table IV, where we present the precision and recall in terms of each sheet, in Figure 3, we show the percentage recall based on each fault type. We discuss the observations as follows.

1) Most Frequently Detected (40-60%): CRP or Constant RePlacement shows the maximum recall rate of detection, possibly because it easily and frequently violates all dominating type of invariants shown in Table V, namely Fixed Value, Range, Upper and Lower Bound, and even Column interrelations and "One of" type. The value of a constant 10 replaced with 100, or 2.22 replaced by 22.2, can largely affect the resulting computation causing invariant violations.

AOR or Arithmetic Operator Replacement, shows a decent recall, as replacing an arithmetic operator with a different one can often change the computed result drastically. Similar to CRP, this tends to cause violations of the dominating type of invariants shown in Table V.

FDL (Formula DeLetion), RFR (ReFerence Replacement), and RCR (Reference for Constant Replacement), all show decent recall rates of detection as they have larger chances of drastically changing the value of a cell, and tend to violate the more frequently inferred type of invariants like Fixed Value.

2) Least Fequently Detected (0-25%): FFR or Formula Function Replacement is a specific case. The scope of applying this mutation is narrow as it can only operate on formulas with functions that can be realistically replaced with another function. In their design of this mutation operator, Abraham and Erwig [6] discuss using `AVERAGE` in place of `SUM`. We believe this is also relevant in case of similarly named functions such as `SUM` and `SUMSQ` (sum of square of the arguments). In our study we could only use these above two replacements. As evident from the results, these faults are not frequently detected. One possible reason is the fact that columns having formulas with `SUM` function do not get covered by invariants such as Fixed Value. They get covered mostly by Lower Bound invariants. As discussed in Section VI-A, Lower Bound invariants, especially those of form $>= 0.0$ are not highly effective in terms of accuracy. Also, evidently they cannot detect the deviations caused by replacing `SUM` with `SUMSQ`, or replacing `SUM` with `AVERAGE`, and vice versa.

FRC or Formula Replacement with Constant deals with replacing a formula with its computed value: essentially what is termed in programming parlance as *hard coding*. From the perspective of value, it does not produce any change. Hence it fails to violate the dominating invariants which mostly are value-based.

LCR (Logical Connector Replacement), ROR (Relational Operator Replacement), CRS (Contiguous Range Shrinking), and CRE (Contiguous Range Expansion), all show drops in recall rates. LCR and ROR tend to be mostly covered by the less effective "One of" type invariants, while CRS and CRE suffer from limited applicability to columns mostly covered by Lower Bound invariants.

3) Applicability of Operators: Apart from the individual characteristics of the faults as discussed above, the applicability of the mutation operators deserves attention. The fault types showing higher recall rates, CRP, AOR, FDL, RFR, and RCR, are all based on operators that are more universally applicable to different types of spreadsheet formulas. This is reflected in Table III. The respective total number of faults injected of these types are 13, 12, 16, 16, and 13. The larger applicability increases the probability of lying in columns that are covered by a larger spectrum of different invariant types, and leads to increased likelihood of detection.

In contrast, the faults showing lowest recall rates, CRS, CRE, LCR, and ROR, are limited in scope of their application. In Table III, their respective numbers are 6, 6, 4, and 6. CRS, and CRE, are only applicable to formulas using ranges, whereas LCR, and ROR are applicable to mostly conditional formulas. As such, the columns in which these types of faults can occur, are less likely to be covered by a large variety of invariants. For example, the conditional formulas mostly are covered by "One of" invariants, which mostly evaluate to a set of permitted values and show least effectiveness in terms of accuracy. FRC is an exception to this trend as it shows low recall despite having better applicability. As discussed in the previous subsection, this is due to the fact that it replaces a formula with its computed value, and as such cannot violate the dominating invariant types which are mostly value-based.

To summarize, as answers to RQ2, we observe that the factors affecting the accuracy of inferred invariants in terms of recall rate of fault detection, are 1) the type of spreadsheet formulas and data, that determine the types of invariants inferred, and 2) the type of faults. Fixed Value type of invariants show most effectiveness in terms of accuracy, while "One of" type invariants appear to be least effective. Faults based on widely applicable mutation operators such as CRP and FDL get more frequently detected. Faults based on operators with limited scope of application such as CRS, CRE, LCR, and ROR, are less frequently detected.

VII. Related Work

The topic of testing and fault detection in spreadsheets has been explored by researchers. Rothermel *et al.* and Fisher *et al.* proposed and improved upon the WYSIWYT approach for spreadsheet testing [23], [24], [25]. Abraham *et al.* proposed another approach for automated test case generation [26] demonstrating improvement compared to [24] using the same experimental setup. In recent times, an approach for spreadsheet fault localization based on spreadsheet smells was proposed by Abreu *et al.* [27].

The concept of invariant-based testing was touched upon by Burnett in their work on assertion-based testing for spreadsheets [8]. Related is also the work by Hermans [21], where the author demonstrates the use of existing "test formulas" in spreadsheets of the EUSES corpus, and proposes an approach for extracting such formulas from spreadsheets to aid users in testing.

VIII. Threats to Validity

A threat to external validity of our results concerns the representativeness of the sheets selected. We argue that the corpora from which they were chosen, both VEnron [4] and the "filby" folder of EUSES [5], are fairly representative of the two types of spreadsheets we were targeting, namely industrial and idealistic, as explained in Section IV-B. Our exploratory

study is a necessary first step towards conducting further large scale generalization of the results through quantitative evaluations.

A second threat to external validity concerns the representativeness of the set of mutation operators used in our fault model (Section IV-C3). An empirical evaluation about how well these operators reflect real-world errors committed by spreadsheet users is not available. However, the operators take into account spreadsheet error types reported in literature [6], and as such, provide us the best systematic alternative to conjuring up faults in an *ad hoc* basis ourselves.

IX. Concluding Remarks and Future Work

The objective of this work was to investigate the effectiveness of automatically inferred invariants, in detecting regression faults in spreadsheets. Results show that the accuracy, in terms of recall rate of fault detection, shows extreme variation from case to case. The variation ranges between 0-88%, with a mean of 37%. The accuracy depends on 1) the type of spreadsheet formulas and data, that determine the types of invariants inferred, and 2) the type of faults.

Based on the outcomes of this study, a first direction for future work is to generalize the results by validating the observations about factors affecting the accuracy in a controlled setting. Another direction is to explore the correlation of spreadsheet smells [28] with the accuracy of inferred invariants, as spreadsheet smells characterize spreadsheet quality. Through these research directions, our ultimate goal is to make the spreadsheet programming paradigm less fault-prone.

References

[1] S. Roy, F. Hermans, and A. van Deursen, "Spreadsheet testing in practice," in *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE, 2017, pp. 338–348.

[2] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao, "The daikon system for dynamic detection of likely invariants," *Science of Computer Programming*, vol. 69, no. 1, pp. 35–45, 2007.

[3] S. Mirshokraie and A. Mesbah, "Jsart: Javascript assertion-based regression testing," *Web Engineering*, pp. 238–252, 2012.

[4] 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 the 38th International Conference on Software Engineering Companion*. ACM, 2016, pp. 162–171.

[5] M. Fisher and G. Rothermel, "The EUSES spreadsheet corpus: a shared resource for supporting experimentation with spreadsheet dependability mechanisms," in *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4. ACM, 2005, pp. 1–5.

[6] R. Abraham and M. Erwig, "Mutation operators for spreadsheets," *IEEE Transactions on Software Engineering*, vol. 35, no. 1, pp. 94–108, Jan 2009.

[7] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, 1969.

[8] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, and C. Wallace, "End-user software engineering with assertions in the spreadsheet paradigm," in *Proc. of ICSE '03*, 2003, pp. 93–103. [Online]. Available: http://dl.acm.org/citation.cfm?id=776816.776828

[9] R. V. Binder, *Testing object-oriented systems: models, patterns, and tools*. Addison-Wesley Professional, 2000.

[10] R. DeMillo, R. Lipton, and F. Sayward, "Program mutation: A new approach to program testing," *Infotech State of the Art Report, Software Testing*, vol. 2, no. 1979, pp. 107–126, 1979.

[11] T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Theoretical and empirical studies on using program mutation to test the functional correctness of programs," in *Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1980, pp. 220–233.

[12] K. N. King and A. J. Offutt, "A fortran language system for mutation-based software testing," *Software: Practice and Experience*, vol. 21, no. 7, pp. 685–718, 1991.

[13] Y.-S. Ma, J. Offutt, and Y. R. Kwon, "Mujava: An automated class mutation system," *Software Testing, Verification and Reliability*, vol. 15, no. 2, pp. 97–133, 2005.

[14] R. R. Panko, "What we know about spreadsheet errors," *Journal of End User Computing*, vol. 10, pp. 15–21, 1998.

[15] R. R. Panko and R. Halverson, "Spreadsheets on trial: A survey of research on spreadsheet risks," in *System Sciences, 1996., Proceedings of the Twenty-Ninth Hawaii International Conference on,*, vol. 2. IEEE, 1996, pp. 326–335.

[16] C. M. Allwood, "Error detection processes in statistical problem solving," *Cognitive science*, vol. 8, no. 4, pp. 413–437, 1984.

[17] S. Roy, F. Hermans, E. Aivaloglou, J. Winter, and A. van Deursen, "Evaluating automatic spreadsheet metadata extraction on a large set of responses from mooc participants," in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, vol. 1. IEEE, 2016, pp. 135–145.

[18] F. Hermans, M. Pinzger, and A. Van Deursen, "Supporting professional spreadsheet users by generating leveled dataflow diagrams," in *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 2011, pp. 451–460.

[19] W. Dou, C. Xu, S. C. Cheung, and J. Wei, "Cacheck: detecting and repairing cell arrays in spreadsheets," *IEEE Transactions on Software Engineering*, vol. 43, no. 3, pp. 226–251, 2017.

[20] L. Xu, W. Dou, C. Gao, J. Wang, J. Wei, H. Zhong, and T. Huang, "Spreadcluster: recovering versioned spreadsheets through similarity-based clustering," in *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*. IEEE, 2017, pp. 158–169.

[21] F. Hermans, "Improving spreadsheet test practices," in *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*. IBM Corp., 2013, pp. 56–69.

[22] G. Filby, *Spreadsheets in Science and Engineering*. Springer Berlin Heidelberg, 2013. [Online]. Available: https://books.google.nl/books?id=-6HvCAAAQBAJ

[23] K. J. Rothermel, C. R. Cook, M. M. Burnett, J. Schonfeld, T. R. Green, and G. Rothermel, "Wysiwyt testing in the spreadsheet paradigm: An empirical evaluation," in *Software Engineering, 2000. Proceedings of the 2000 International Conference on*. IEEE, 2000, pp. 230–239.

[24] M. Fisher, M. Cao, G. Rothermel, C. R. Cook, and M. M. Burnett, "Automated test case generation for spreadsheets," in *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*. IEEE, 2002, pp. 141–151.

[25] M. Fisher II, G. Rothermel, D. Brown, M. Cao, C. Cook, and M. Burnett, "Integrating automated test generation into the wysiwyt spreadsheet testing methodology," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 15, no. 2, pp. 150–194, 2006.

[26] R. Abraham and M. Erwig, "Autotest: A tool for automatic test case generation in spreadsheets," in *Visual Languages and Human-Centric Computing (VL/HCC'06)*. IEEE, 2006, pp. 43–50.

[27] R. Abreu, J. Cunha, J. P. Fernandes, P. Martins, A. Perez, and J. Saraiva, "Smelling faults in spreadsheets," in *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 2014, pp. 111–120.

[28] F. Hermans, M. Pinzger, and A. van Deursen, "Detecting code smells in spreadsheet formulas," in *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 2012, pp. 409–418.

TUD-SERG-2018-002
ISSN 1872-5392