Mutation-based spreadsheet debugging

Birgit Hofer and Franz Wotawa, Institute for Software Technology,
{bhofer,wotawa}@ist.tugraz.at

25.11.2013

Problem statement

- Given a *spreadsheet* and indications about its misbehavior, i.e., *expected cells' values*.
- Search for the root causes and potential corrections using mutation operators.
- Ideas based on previous work on program debugging based on mutations / genetic programming:
  - W. Weimer, S. Forrest, C. Le Goues, and T. Nguyen, *Automatic program repair with evolutionary computation*, Communications of the ACM, vol. 53, no. 5, pp. 109–116, 2010.

Motivating example

- Example from economy (Reinhart and Rogoff, Roosevelt Institute)!
- Fault in spreadsheet lead to wrong conclusions

	A	B	I	J	K	L	M
2			Real GDP growth				
3			Debt/GDP				
4	Country	Coverage	30 or less	30 to 60	60 to 90	90 or above	30 or less
26			3,7	3,0	3,5	1,7	5,5
27	Minimum		1,6	0,3	1,3	-1,8	0,8
28	Maximum		5,4	4,9	10,2	3,6	13,3
29							
30	US	1946-2009	n.a.	3,4	3,3	-2,0	n.a.
31	UK	1946-2009	n.a.	2,4	2,5	2,4	n.a.
32	Sweden	1946-2009	3,6	2,9	2,7	n.a.	6,3
33	Spain	1946-2009	1,5	3,4	4,2	n.a.	9,9
34	Portugal	1952-2009	4,8	2,5	0,3	n.a.	7,9
35	New Zealand	1948-2009	2,5	2,9	3,9	-7,9	2,6
36	Netherlands	1956-2009	4,1	2,7	1,1	n.a.	6,4
37	Norway	1947-2009	3,4	5,1	n.a.	n.a.	5,4
38	Japan	1946-2009	7,0	4,0	1,0	0,7	7,0
39	Italy	1951-2009	5,4	2,1	1,8	1,0	5,6
40	Ireland	1948-2009	4,4	4,5	4,0	2,4	2,9
41	Greece	1970-2009	4,0	0,3	2,7	2,9	13,3
42	Germany	1946-2009	3,9	0,9	n.a.	n.a.	3,2
43	France	1946-2022	4,9	2,7	3,0	n.a.	5,2
44	Finland	1946-2023	3,8	2,4	5,5	n.a.	7,0
45	Denmark	1946-2024	3,5	1,7	2,4	n.a.	5,6
46	Canada	1946-2025	1,9	3,6	4,1	n.a.	2,2
47	Belgium	1946-2026	n.a.	4,2	3,1	2,6	n.a.
48	Austria	1946-2027	5,2	3,3	-3,8	n.a.	5,7
49	Australia	1946-2028	3,2	4,9	4,0	n.a.	5,9
50							
51			4,1	2,8	2,8	=AVERAGE(L30:L44)	
52							

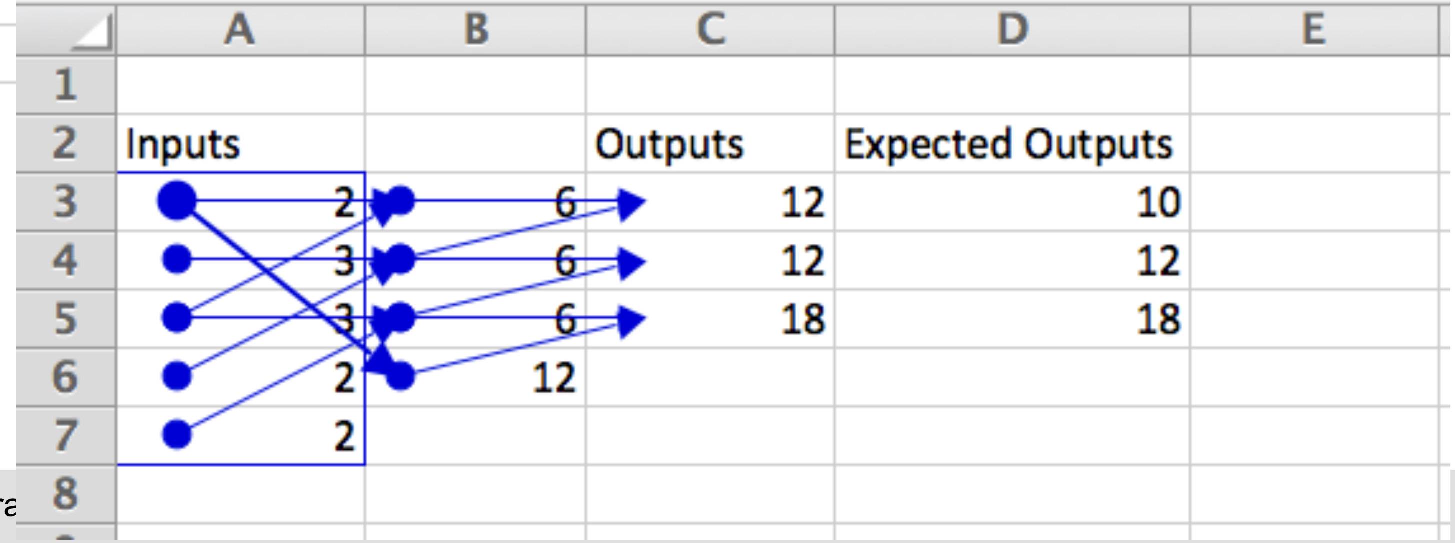
Past work

- GoalDebug [Abraham and Erwig; 2007] – Ranked list of repair suggestions
- Model-based approaches [Jannach and Engler;] and [Hofer et al.; 2013]
- Trace-based approaches [Ruthruff et al.; 2003] – Similar to Tarantula,...

Past work [Hofer et al.; 2013]

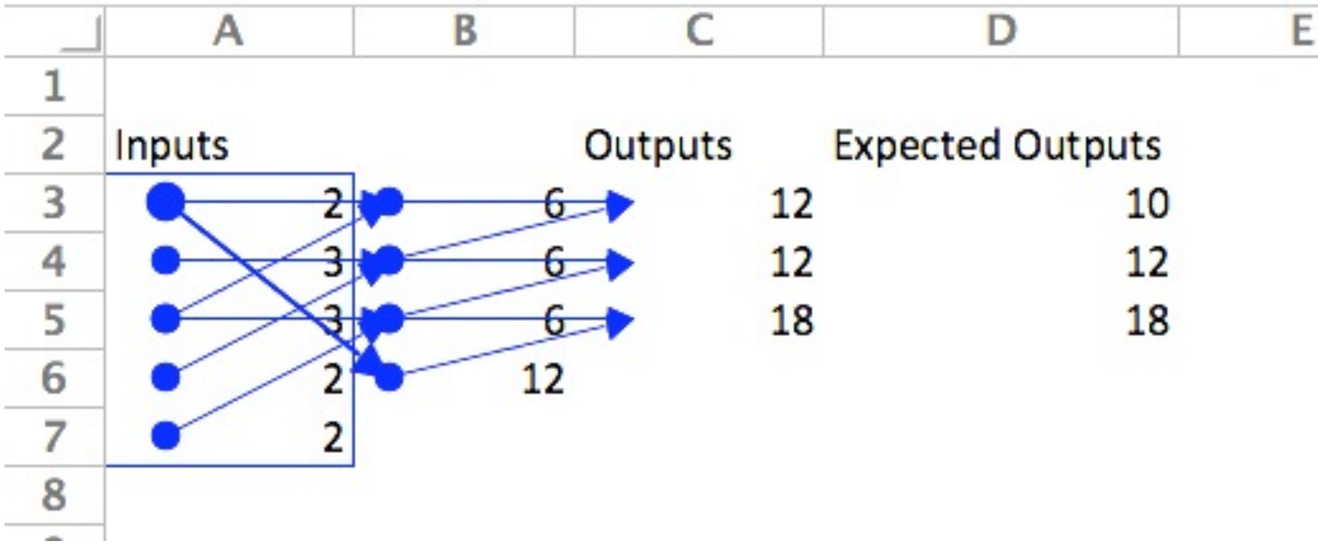
- Have a look at the following simple example

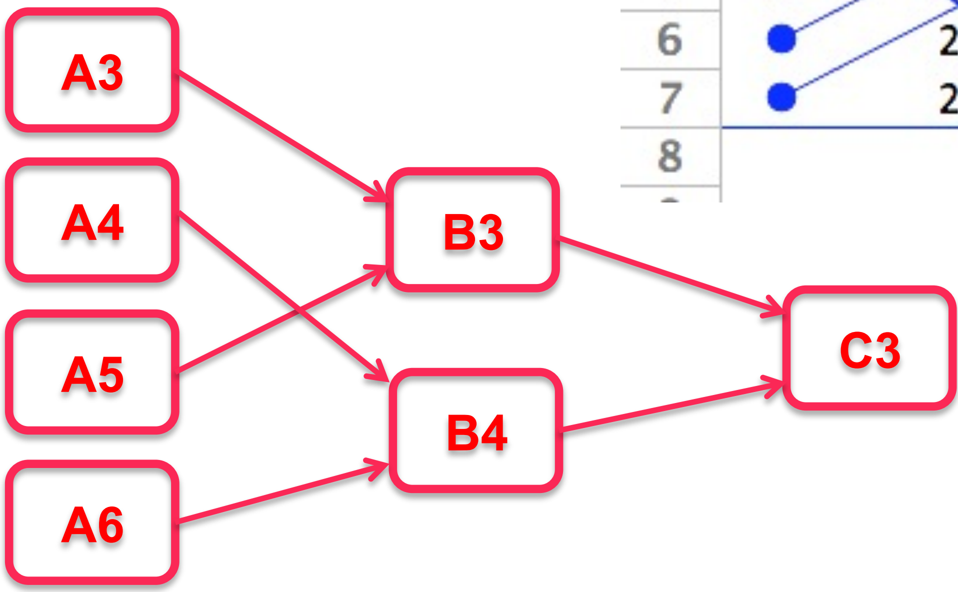
	A	B	C	D
1				
2	Inputs		Outputs	Expected Outputs
3	2	=A3*A5	=B3+B4	10
4	3	=A4*A6	=B4+B5	12
5	3	=A7*A5	=B6+B5	18
6	2	=SUMME(A3:A7)		
7	2			
8				



Cones / Slicing / SFL / etc.

Each cone is a slice and considered as a "trace" (for SFL)





Birgit Hofer and Franz Wotawa, Institute for Software Technology, {bhofer,wotawa}@ist.tugraz.at

Model-based approaches

Convert formulas to constraints and add special predicates AB(C)

	A	B	C	D
1				
2	Inputs		Outputs	Expected Outputs
3	2	=A3*A5	=B3+B4	10
4	3	=A4*A6	=B4+B5	12
5	3	=A7*A5	=B6+B5	18
6	2	=SUMME(A3:A7)		
7	2			
8				

AB(B2) or (B2=A3*A5)

AB(B3) or (B3=A4*A6)

....

Choose AB(C) such that no contradiction occurs anymore!

Obtained results

Characteristic	Avg	Min	Max	Std.dev	Median
Number of formulas	225.0	6	4,170	384.9	104.5
Number of incorrect output cells	1.7	1	22	1.9	1
Number of correct output cells	64.9	0	2,962	162.5	24

Table 1. Characteristics of the created mutants.

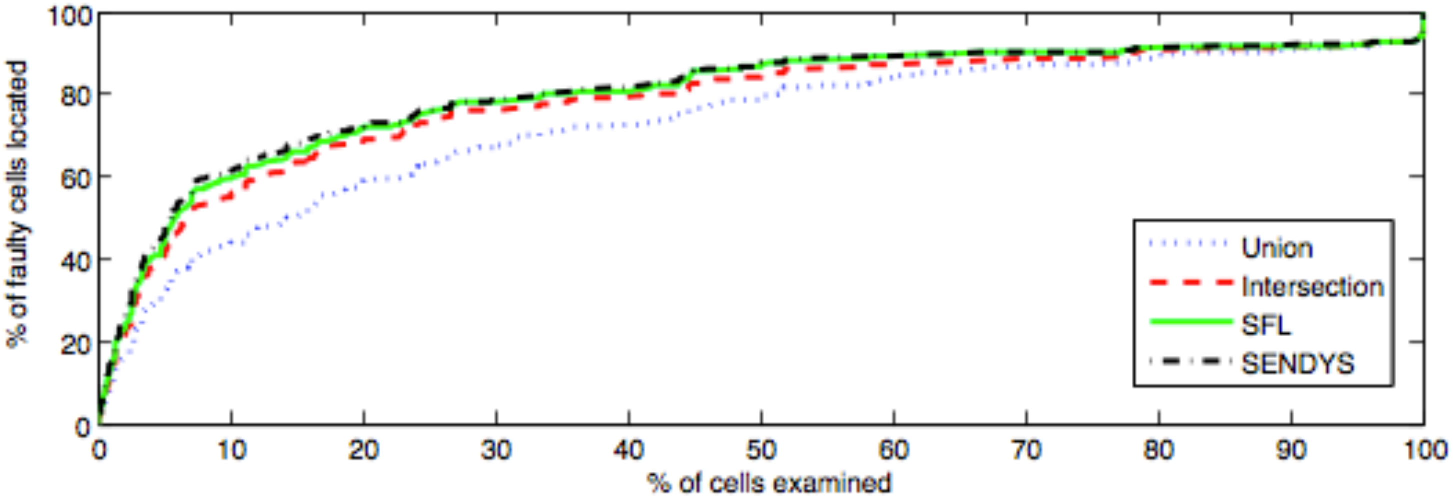


Fig. 1. Comparison of the SFL, SENDYS, the Union and Intersection of the cones in terms of the amount of formula cells that must be investigated.

Obained results (cont)

Technique	Ø absolute ranking	Ø relative ranking	Ø computation time (in ms)
Union (cones of faulty output)	34.8	29.3 %	14.0
Intersection (cones of faulty output)	33.6	27.5 %	13.9
SFL	32.9	27.1 %	15.0
SENDYS	31.9	27.0 %	63.9
CONBUG	33.9	27.9 %	631.7

Table 3. Average ranking and computation time of union, intersection, SFL, SENDYS, and CONBUG. The column 'Ø relative ranking' shows the average ranking of the statement normalized to the number of formulas per spreadsheet. This evaluation comprises 227 spreadsheets.

Research question: ConBug "better" than SENDYS when used for imperative languages. Why is this not the case in the spreadsheet domain?

Mutation-based diagnosis

Short description

3 parts:

1. use cones for focusing on relevant statements
2. generate mutations for relevant statements
3. check correctness (thus eliminating candidates)

ad 1. cones:

$$\textsc{Cone}(c) = c \cup \bigcup_{c' \in \textsc{Formula}(c)} \textsc{Cone}(c')$$

Mutations

ad 2. mutations:

- Constants
  - Change Boolean values
  - Permutate digits
  - Increase or decrease number by 1 – Use a random number instead
  - Change the sign
- References
  - Randomly increase/decrease the borders of areas
  - Randomly change a single reference • Formulas
  - Replace a binary operator (e.g., '+') with another binary operator (e.g., '-')
  - Replace a formula (e.g., 'SUM') with another formula which can process the same arguments (e.g., 'AVG')
  - Remove parts of a formula (e.g., 'A3+A4*3'→'A3*3')
  - Relocate parts of a formula (e.g., 'if(A1 > 1; A2; A3)'→'if (A1 > 1; A3; A2)')

Mutations (cont)

- Crossover not considered
- Fitness function (ako ad 3. checking):

$$\text{FITNESS}(m) = -\left(\omega_{\text{initial}} \cdot |M \cap O| + \omega_{\text{new}} \cdot |M \backslash O|\right)$$

Output cells of mutants with wrong values

Output cells with wrong values

Throw away all mutants where $\text{FITNESS}(m) < -\omega_{\text{initial}} \cdot |O|$

The mutation-based diagnosis algorithm

- computes new mutants until results is found, or
- limit is reached!

Algorithm 1 Evolutionary algorithm

```
procedure EVOLUTION(s, o)    ▷ s: Spreadsheet, o: set of (cell, expectedOutput) pairs
    cones ← {}
    for (cell, expectedOutput) ∈ o do
        if value(cell) ≠ expectedOutput then
            Cones ← Cones ∪ CONE(cell)
        end if
    end for
    i ← 0                                      ▷ Generation counter
    P ← {m}                                    ▷ Population
    while i < maxGenerations do
        G ← {}                                 ▷ New generation
        j ← 0
        while j < populationSize do
            Randomly select c from cones
            Randomly select m₁ from P
            m₂ ← MUTATE(m₁, c)
            if FITNESS(m₂) = 0 then
                return m₂                      ▷ Solution found
            end if
            if FITNESS(m₂) ≥FITNESS(m) then
                G ← G ∪ {m₂}                   ▷ Improved mutant
            end if
            j = j + 1
        end while
        P ← P ∪ G
        i = i + 1
    end while
    return FAIL
end procedure
```

Empirical results

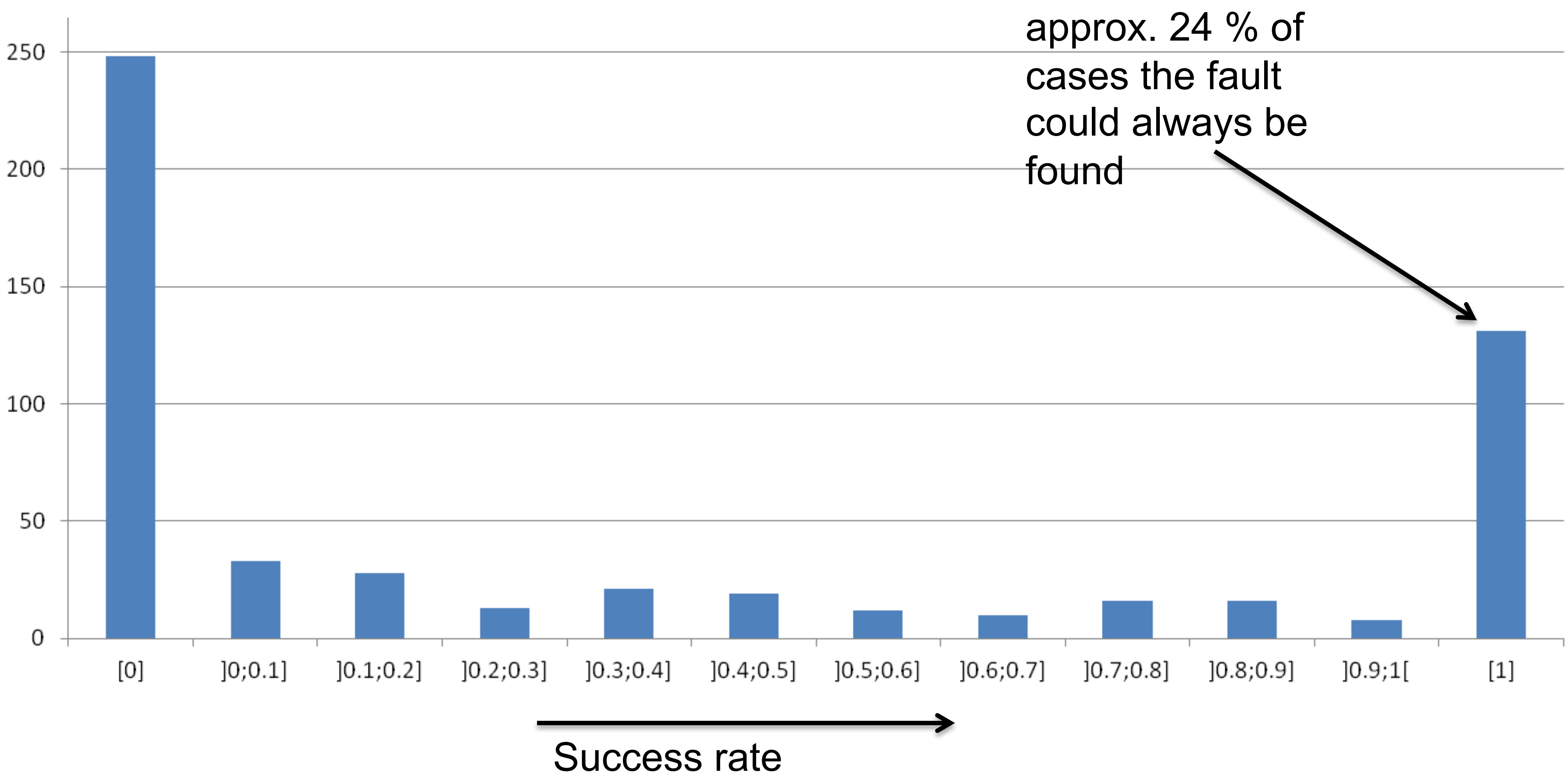
Preliminaries:

- EUSES spreadsheet corpus
- Remove some spreadsheets (no input cells, too small,..). Approx. 500 spreadsheets remained (with 6..4,000 formulas, and on avg. 225).
- Both weights set to 1
- 16 trails per spreadsheet

Obtained results:

- In about 55 % of the cases the fault has been located and corrected (at least once)
- Computation time ranged between 2 ms and 40 minutes (avg. 16.3 seconds)

Empirical results (cont)



Conclusions

- Presented a mutation-based approach for fault localization and correction for spreadsheets
- Solution = Mutation that explains faulty behavior
- Not always successful (like the other mutation-based approaches)

Thank you for your attention!

Birgit Hofer and Franz Wotawa, Institute for Software Technology, {bhofer,wotawa}@ist.tugraz.at

25.11.2013