Fault Localization for Spreadsheets

Elisabeth Getzner

November 13, 2014

Introduction

Fault Localization in Spreadsheets

Aids the user in finding the location of the fault (faulty cell) provided the spreadsheet shows erroneous behavior.

Goals

- Reduce the search space (less cells)
- Prioritize the search (which cell should we inspect first)

Approaches

Trace- and Model-based approaches, Combination approaches

Fault Localization in Spreadsheets

Trace-based Approaches

- WYSIWYT
  - Test Count
  - Blocking Technique
  - Nearest Consumer
- SFL
- MOSTINFLUENTIAL

Model-based Approaches

- CONBUG
- EXQUISITE

Combined Approaches

- SENDYS
- WYSIWYT and UCHECK

Comparison Criteria

- User Input
  - Complexity and size
- Algorithm
  - Underlying concepts
  - Runtime and fault complexity
- Output
  - Type and form
- Evaluation
  - Metric used to measure the success

Running Example

	A	B	C	D	E
1		Hours	Salary	Bonus	Sum
2	Jones	17	272	26	298
3	Smith	13	208	0	208
4	Rogers	20	320	40	360
5	Total		800	66	866

Figure 1: Value view with fault in D2.

	A	B	C	D	E
1		Hours	Salary	Bonus	Sum
2	Jones	17	=B2*16	=IF(B2>15, C3 /8,0)	=SUM(C2:D2)
3	Smith	13	=B3*16	=IF(B3>15, C3 /8,0)	=SUM(C3:D3)
4	Rogers	20	=B4*16	=IF(B4>15, C4 /8,0)	=SUM(C4:D4)
5	Total		=SUM(C2:C4)	=SUM(D2:D4)	=SUM(E2:E4)

Figure 2: Formula view, with D2 referencing C3 instead of C2.

Trace-based Approaches

User Input for trace-based approaches

User Input: Testing Decisions (TD = $TD^+ \cup TD^-$)

- (✓ /✗) for each cell, judging values
- Advanced: assertions, multiple input values

Algorithm Input: Test Cases (TC)

- Slice or CONE for each TD
- $TD^+ \rightarrow \mathrm{TC_P}$, $TD^- \rightarrow \mathrm{TC_F}$
- Reduce search space to relevant cells
- Dynamic vs. static

Issues:

- oracle mistakes
- coincidental correctness

User Input and Test Cases

	A	B	C	D	E
1		Hours	Salary	Bonus	Sum
2	Jones	17	272	26	298
3	Smith	13	208	0	✓ 208
4	Rogers	20	320	40	360
5	Total		✓ 800	66	✗ 866

Figure 3: Test case visualization, with testing decisions marked with ✓ /✗ .

Blocking

- Based on Dicing
  - Exclude any cell that participates in passing test case
- "Very Low" fault likelihood for those cells
- Relies on correct testing decisions
  (coincidental correctness or oracle mistakes)
- Count only "reachable" test cases:

$$FL_{BL}(c) = \max(1\ ,\ 2 \cdot |\,TC_{F\ ,U}(c)| - |\,TC_{P\ ,U}(c)|),$$

with $TC_{F\ ,U}(c)$ the number of unblocked failed test cases

Blocking Example

	A	B	C	D	E
1		**Hours**	**Salary**	**Bonus**	**Sum**
2	Jones	17	272	26	298
3	Smith	13	208	0	208
4	Rogers	20	320	40	360
5	**Total**		*800*	*66*	✗ *866*

Figure 4: Failing test case originating from E5

Blocking Example (2)

	A	B	C	D	E
1		Hours	Salary	Bonus	Sum
2	Jones	17	272	26	298
3	Smith	13	208	0	208
4	Rogers	20	320	40	360
5	**Total**		☑ *800*	*66*	☒ *866*

Figure 5: Passing test case originating from C5

Blocking Example (3)

	A	B	C	D	E
1		Hours	Salary	Bonus	Sum
2	Jones	17	272	26	298
3	Smith	13	208	0	☑ 208
4	Rogers	20	320	40	360
5	**Total**		☑ *800*	*66*	☒ *866*

Figure 6: Passing test case originating from E3

Output

- Likelihood value for all relevant cells
  - Low likelihoods with few test cases
- **Tie:** multiple cells with the same likelihood
- Ranking

	A	B	C	D	E
1		**Hours**	**Salary**	**Bonus**	**Sum**
2	Jones	17	1	2	2
3	Smith	13	0.1	0.1	0.1
4	Rogers	20	1	2	2
5	**Total**		0	0	2

Figure 7: Heat Map output for Blocking

Metrics

WYSIWYT

measures visual effectiveness: $\text{eff}_{\text{FL}} = \text{avg}_{\text{FL}}(C_F) - \text{avg}_{\text{FL}}(C_{NF})$

- no info on position of fault (first ranked)
- too much weight given to lower ranked non-faulty cells

SFL

absolute and relative rank (= absolute rank / $|C|$) of the faulty cell

- single faults only!
- critical tie (contains faulty cell) needs to be considered

Which cells are in C

- all formula cells,
- all cells $FL > 0$,
- cells ranked higher than the faulty cell

Multiple Fault Complexity

- limited trace-based support
- not supported by output

Figure 8: Single fault

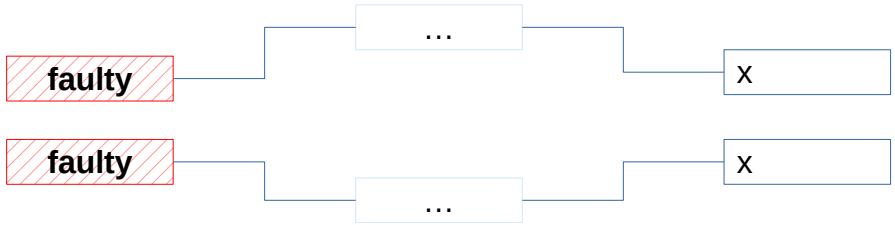

Figure 9: Independent double fault

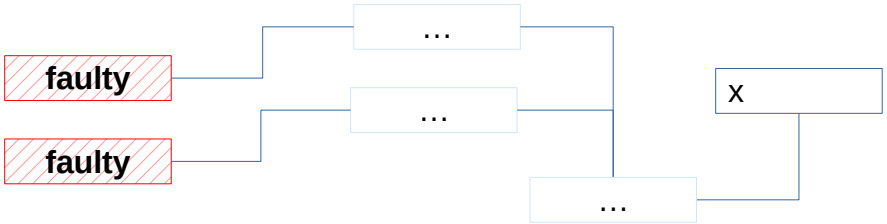

Figure 10: Dependent double fault

Figure 11: Nested double fault

Independent Faults - Example

	A	B	C	D	E
1		Hours	Salary	Bonus	Sum
2	Jones	17	=B2*16	=IF(B2>15, C3 /8,0)	=SUM(C2:D2)
3	Smith	13	=B3*16	=IF(B3>15, C4 /8,0)	=SUM(C3:D3)
4	Rogers	20	=B4*16	=IF(B4>15, C5 /8,0)	=SUM(C4:D4)
5	Total		=SUM(C2:C4)	***=SUM(D2:D4)***	***=SUM(E2:E4)***

Figure 12: Formula view, multiple faults `D2:D4` highlighted

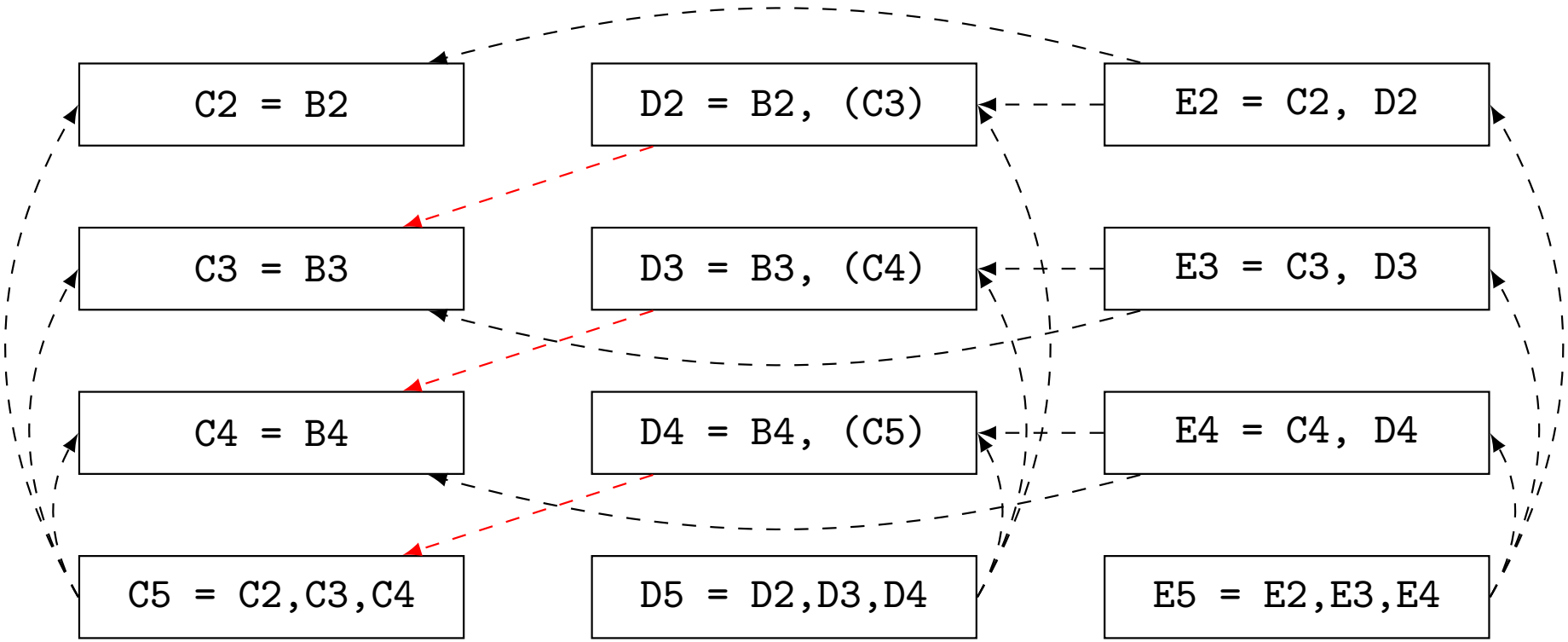

Figure 13: Dependency graph for the multiple fault example

	A	B	C	D	E
1		Hours	Salary	Bonus	Sum
2	Jones	17	272	26	✗ 298
3	Smith	13	208	0	208
4	Rogers	20	320	100	✗ 420
5	**Total**		800	*126*	*926*

Figure 14: Testing decisions for the multiple fault example

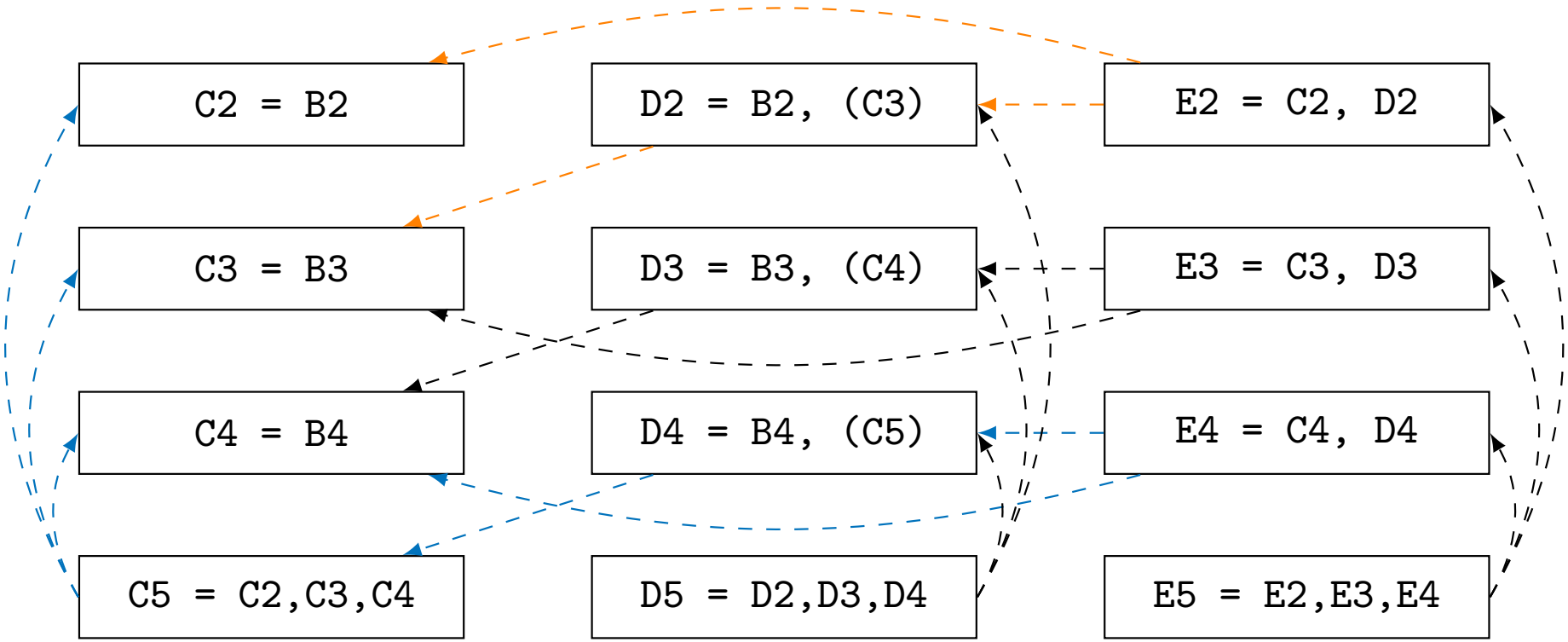

Figure 15: Dependency graph indicating test case creation

	A	B	C	D	E
1		Hours	Salary	Bonus	Sum
2	Jones	17	272	26 ✗	298
3	Smith	13	208	0	208
4	Rogers	20	320	100 ✗	420
5	Total		800	*126*	*926*

Figure 16: Testing decisions for the multiple fault example - independent faults

- Cells that participate in most failing test cases rank highest
  - Non-faulty cells might receive higher rank than faulty
- Decrease in confidence
- Could be detected if intersection is empty

Strengths and Weaknesses

Strengths

(+) Only (✓ /✗) required as input
(+) Low runtime requirements
(+) Intuitive output

Weaknesses

(-) Lacking support for multiple faults
(-) Often dependent on many test cases
(-) Little reduction in search space (only prioritization)

Other Approaches

Model-based Debugging

- Model (spreadsheet) and description (user input) allows information on conflicting cells.
- Solvers are used to check consistency - higher runtime
- Input
  - Test cases with expected output values (✗ not enough!)
  - Assumes perfect oracle
  - Additional assertions possible
- Output
  - Diagnosis: set of one or more cells that explain the fault (multiple faults)
  - Set of diagnoses - difficult to prioritize
  - No ranking - no prioritization
  - Reduction of the search space by excluding cells

SENDYS

Spectrum ENhanced Dynamic Slicing

- Combines SFL with lightweight model-based approach (CONEs = conflicts, hitting sets = diagnoses)
- Allows combination of
  - Ranking/likelihoods for single cells (trace-based) with
  - Diagnoses of potentially multiple cardinality (model-based)
- More diagnoses than model-based (no solver calls - no explanations)
- Similar issues with multiple faults as trace-based approaches

Comparison

Comparison

Table 1:Comparison of the discussed approaches

Approach:	Trace-based	Model-based	SENDYS
User Input			
Testing decisions	value (✓ /✗)	expected value	value (✓ /✗)
Required complexity	low	high	low
Optional complexity	high	very high	low
Algorithm			
Based on	heuristics	explanations	heuristics, diagnoses
Fault complexity	mostly single	multiple	multiple
Runtime complexity	low	high	low-moderate
Output			
Type	ranking	diagnoses	ranking, diagnoses

Thank You

Any Questions?