Selected Topics of Software Technology 3
Model-based Software Debugging
Part III

Birgit Hofer
Institute for Software Technology
Outline

- Model-based debugging for Spreadsheets
  - Dependency-based models
- Model-based software debugging for Java programs
  - SSA form
  - Loop unrolling
  - Path modelling
  - Combined approaches
Outline

- Model-based debugging for Spreadsheets
  - Dependency-based models
- Model-based software debugging for Java programs
  - SSA form
  - Loop unrolling
  - Path modelling
  - Combined approaches
Repetition: Automatic Reasoning + Types of models

System description SD:
- \( AB(D2) \vee D2 = B2 + C2 \)
- \( AB(D3) \vee D3 = D4 / D2 \)
- \( AB(B4) \vee B4 = B3 \times B2 \)

Test case OBS:
- \( D7 == 0.786 \)
- \( B7 == 0.750 \)

For single faults:
- \( \sum(AB(D2), AB(D3), \ldots) == 1 \)

Should be 78.6%

More General:
- \( AB(D2) \vee \text{behavior}(D2) \)
- \( AB(D3) \vee \text{behavior}(D3) \)
- \( AB(B4) \vee \text{behavior}(B4) \)

Single Fault:
- \( \{D5\} \)
- \( \{D6\} \)
- \( \{D7\} \)

Double Fault:
- \( \{D3,D4\} \)

Spreadsheet + Expect. Output → Conversion → Constraints (Model) → Solve → Diagnoses
Repetition: Different Types of Models

How can we model the behavior of the formulas?

What are the pros and cons of these models?
Repetition: Models for a Spreadsheet’s Behavior

**Value-based**
- $D2 == B2 + C2$
- $D3 == D4 / D2$
- $B4 == B3 \times B2$

+ exact, few diagnoses
- computation time
- Reals: lacking support

**Dependency-based**
- $ok(B2) \land ok(C2) \rightarrow ok(D2)$
- $ok(D4) \land ok(D2) \rightarrow ok(D3)$
- $ok(B3) \land ok(B2) \rightarrow ok(B4)$

+ fast
+ only Boolean
- many diagnoses
Repetition: Different Types of Models

How can we derive a dependency-based model from a value-based model?
Example II – Continuation

- Replace all discrete variables with Boolean variables

- Replace the test case values with correctness information
  - All cells with input values are true
  - All cells with correct output values are true
  - All cells with incorrect output values are false

- Replace all formula constraints with imply-constraints, e.g. \( \text{AND}(B2,B3) \rightarrow B4 \)
  - You can use the Minion Constraint \( \text{min}([\text{var1, var2, ...}], \text{result}) \) to model ‘AND’
Example II – Continuation II

- Compute the single fault diagnoses for this new (dependency-based) model and compare the results with those of the value-based model!

- Explain the reason for the difference!

- Do you have any ideas how to improve the dependency-based model?
Example II
Reason for difference
Improving the Dependency-based Model

- Use $\leftrightarrow$ instead of $\rightarrow$ (wherever possible)
  - $\text{ok}(B2) \land \text{ok}(C2) \leftrightarrow \text{ok}(D2)$
  - $\text{ok}(D4) \land \text{ok}(D2) \leftrightarrow \text{ok}(D3)$
  - $\text{ok}(B3) \land \text{ok}(B2) \leftrightarrow \text{ok}(B4)$

Not possible in case of

- Coincidental correctness
  - Conditional like IF-function
  - Abstraction function like MIN, MAX, COUNT
  - Boolean
  - Multiplication by zero
  - Power with 0 or 1 as base number or 0 as exponent
Example II – Continuation III

- Improve the dependency-based model and compute the diagnoses!

- Compare the results to the result of the other models!
Empirical Evaluation

- Java implementation using
  - Apache POI
  - Minion Constraint solver
- Spreadsheets from Integer corpus
  - Single faults only

- 94 spreadsheets
- 63 spreadsheets
- 31 spreadsheets → Timeout (20 minutes) for 31 spreadsheets for Value-based model
## Empirical Evaluation

<table>
<thead>
<tr>
<th>Model</th>
<th>63 spreadsheets</th>
<th>31 spreadsheets</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of single fault diagnoses</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value-based</td>
<td>4.0</td>
<td>-</td>
</tr>
<tr>
<td>Dependency-based</td>
<td>13.2</td>
<td>45.0</td>
</tr>
<tr>
<td>Improved Dep.-based</td>
<td>11.0</td>
<td>38.6</td>
</tr>
<tr>
<td><strong>Runtime in ms</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value-based</td>
<td>56,818.8</td>
<td>&gt; 20 minutes</td>
</tr>
<tr>
<td>Dependency-based</td>
<td>32.0</td>
<td>187.4</td>
</tr>
<tr>
<td>Improved Dep.-based</td>
<td>31.6</td>
<td>164.8</td>
</tr>
</tbody>
</table>
Comparison of the different models

Spreadsheet + Expect. Output → Conversion → Constraints (Model) → Solve → Diagnoses

**AB(D2) ∨ behavior(D2)**
**AB(D3) ∨ behavior(D3)**
**AB(B4) ∨ behavior(B4)**

**Value-based (VB)**
D2 = B2 + C2

- less diagnoses
- high computation time
  - 1/3: 20 min timeout
  - 2/3: 1 minute

**Dependency-based (DB)**
ok(B2) ∧ ok(C2) → ok(D2)

- many diagnoses
- 3.3 times more than VB
- low computation time
  - less than 1 second

**Improved dep.-based (IDB)**
ok(B2) ∧ ok(C2) ↔ ok(D2)

- reduced number of diagnoses
- 15% less diagnoses than DB
- low computation time
  - less than 1 second

**Coincidental correctness**
- Conditional like IF-function
- Abstraction functions
- Boolean
- Multiplication and Power
Outline

- Model-based debugging for Spreadsheets
  - Dependency-based models

- Model-based software debugging for Java programs
  - SSA form
  - Loop unrolling
  - Path modelling
  - Combined approaches
Example

1. `public int[] getTastes(red, blue, green, yellow) {`
2. `red = red * 5;`
3. `sweet = 2 * red * green; // FAULT`
4. `sour = 0;`
5. `i = 0;`
6. `while(i < red) {
7.       sour = sour + green;
8.       i = i + 1;
9.    }
10. salty = blue + yellow;
11. yellow = sour + 1;
12. bitter = 2 * yellow + green; // FAULT`
13. `return {bitter, sweet, sour, salty};`
14. `}

Source: Unravel project (http://hissa.nist.gov/unravel/)
Solution

red = red * 5;
sweet = 2 * red * green;
sour = 0;

red1 = red0 * 5;
sweet1 = 2 * red1 * green0;
sour1 = 0;

SSA form
- Static single assignment form
- Unique name for each redefinition of a variable
red = red * 5;
sweet = 2 * red * green;
sour = 0;

red1 = red0 * 5;
sweet1 = 2 * red1 * green0;
sour1 = 0;

AB2 v red1 == red0 * 5;
AB3 v sweet1 == 2 * red1 * green0;
AB4 v sour1 == 0;
Minion constraints

\[
\begin{align*}
AB2 \lor \text{red1} &= \text{red0} \times 5; \\
AB3 \lor \text{sweet1} &= 2 \times \text{red1} \times \text{green0}; \\
AB4 \lor \text{sour1} &= 0;
\end{align*}
\]

\[
\begin{align*}
\text{watched-or}(&\{\text{AB2, product(red_0,5,tmp_var_1)}\}) \\
\text{watched-or}(&\{\text{AB2, eq(red_1,tmp_var_1)}\}) \\
\text{watched-or}(&\{\text{AB3, product(2, red_1,tmp_var_2)}\}) \\
\text{watched-or}(&\{\text{AB3, product(tmp_var_2,green_0,tmp_var_3)}\}) \\
\text{watched-or}(&\{\text{AB3, eq(sweet_0,tmp_var_3)}\}) \\
\text{watched-or}(&\{\text{AB4, eq(sour_1,0)}\})
\end{align*}
\]
\texttt{Execution Trace}
\begin{align*}
\text{red} &= \text{red} \times 5; \\
\text{sweet} &= 2 \times \text{red} \times \text{green}; \\
\text{sour} &= 0;
\end{align*}

\texttt{SSA conv. Exec. Trace}
\begin{align*}
\text{red}_1 &= \text{red}_0 \times 5; \\
\text{sweet}_1 &= 2 \times \text{red}_1 \times \text{green}_0; \\
\text{sour}_1 &= 0;
\end{align*}

\texttt{Constraints}
\begin{align*}
\text{AB2} \vee \text{red}_1 &= \text{red}_0 \times 5; \\
\text{AB3} \vee \text{sweet}_1 &= 2 \times \text{red}_1 \times \text{green}_0; \\
\text{AB4} \vee \text{sour}_1 &= 0;
\end{align*}

\texttt{Constraint Solver}

find all solutions

\texttt{Diagnoses}

\text{e.g.} \{13\}
Outline

- Model-based debugging for Spreadsheets
  - Dependency-based models
- Model-based software debugging for Java programs
  - SSA form
  - Loop unrolling
  - Path modelling
  - Combined approaches
Loop unrolling

1. static public int test1() {
2.     int a, b, c, d, i;
3.     i = 1;
4.     a = 1;
5.     b = 2;
6.     c = 3;
7.     d = 4;
8.     while (i < 3) {
9.         a = b;
10.        b = c;
11.       c = d;
12.       i++;
13.     }
14.     return a;
15. }

Execution trace

1. i = 1;
2. a = 1;
3. b = 2;
4. c = 3;
5. d = 4;
6. while (i < 3) // True
7. a = b;
8. b = c;
9. c = d;
10. i++;
11. while (i < 3) // True
12. a = b;
13. b = c;
14. c = d;
15. i++;
16. while (i < 3) // False
17. return a;
Outline

- Model-based debugging for Spreadsheets
  - Dependency-based models

- Model-based software debugging for Java programs
  - SSA form
  - Loop unrolling
  - Path modelling
  - Combined approaches
Triangle Example

1. public String getTriangleType(int in1, int in2, int in3){
2.     a = in1;
3.     b = in2;
4.     c = in3;
5.     if (!((a<0) && (b>0) && (c>0))) // condA
6.         return “no triangle”;  
7.     else if ((a+b<c) || (a+c<b) || (b+c<a)) // condB
8.         return “invalid”;
9.     else if (a == b && a == c) // condC
10.        return “equilateral”;  
11.     else if (a == b || a == c || b == c) // condD
12.        return “isosceles”;  
13.     else return “scalene”;  
14. }

Test Case:
    in1 = 2
    in2 = 2
    in3 = 3
    out = “isosceles”
What happens if a fault causes changes in the execution path?

- \(a = \text{in1}\)
- \(b = \text{in2}\)
- \(c = \text{in3}\)

- **condA**
  - then
    - Out = “No triangle”
  - else

- **condB**
  - then
    - Out = “INVALID”
  - else

- **condC**
  - then
    - Out = “equilateral”
  - else

- **condD**
  - then
    - Out = “isosceles”
  - else
    - Out = “scalene”
Modelling

- \( a = \text{in1} \)
- \( b = \text{in2} \)
- \( c = \text{in3} \)
- \( \neg \text{intrace}_0 \lor \text{AB}_1 \lor a_0 = \text{in1} \)
- \( \neg \text{intrace}_0 \lor \text{AB}_2 \lor b_0 = \text{in2} \)
- \( \neg \text{intrace}_0 \lor \text{AB}_3 \lor c_0 = \text{in3} \)

- \( \neg \text{intrace}_0 \rightarrow \neg \text{intrace}_1 \)
- \( \text{AB}_4 \rightarrow \neg \text{intrace}_1 \)
- \( \neg \text{intrace}_0 \lor \text{AB}_4 \lor \neg[(a_0<0) \land (b_0>0) \land (c_0>0)] \leftrightarrow \text{intrace}_1 \)

- \( \neg \text{intrace}_1 \lor \text{AB}_5 \lor \text{output}_0 = \text{“No triangle”} \)
Modelling

\[ \neg \text{intrace}_0 \lor AB_1 \lor a_0 = \text{in1} \]
\[ \neg \text{intrace}_0 \lor AB_2 \lor b_0 = \text{in2} \]
\[ \neg \text{intrace}_0 \lor AB_3 \lor c_0 = \text{in3} \]

\[ \neg \text{intrace}_0 \rightarrow \neg \text{intrace}_1 \]
\[ AB_4 \rightarrow \neg \text{intrace}_1 \]
\[ \neg \text{intrace}_0 \lor AB_4 \lor \neg[(a_0<0) \land (b_0>0) \land (c_0>0)] \leftrightarrow \text{intrace}_1 \]

\[ \neg \text{intrace}_1 \lor AB_5 \lor \text{output}_0 = \text{"No triangle"} \]

\[ \text{in1} = 2 \]
\[ \text{in2} = 2 \]
\[ \text{in3} = 3 \]
\[ \text{output}_0 = \text{"isosceles"} \]
\[ \text{intrace}_0 \]
Modelling

\[ \neg \text{intrace}_0 \lor AB_1 \lor a_0 = \text{in1} \]
\[ \neg \text{intrace}_0 \lor AB_2 \lor b_0 = \text{in2} \]
\[ \neg \text{intrace}_0 \lor AB_3 \lor c_0 = \text{in3} \]

\[ \neg \text{intrace}_0 \rightarrow \neg \text{intrace}_1 \]
\[ AB_4 \rightarrow \neg \text{intrace}_1 \]
\[ \neg \text{intrace}_0 \lor AB_4 \lor \neg[(a_0 < 0) \land (b_0 > 0) \land (c_0 > 0)] \leftrightarrow \text{intrace}_1 \]

\[ \neg \text{intrace}_1 \lor AB_5 \lor \text{output}_0 = \text{"No triangle"} \]

<table>
<thead>
<tr>
<th>in1</th>
<th>= 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>in2</td>
<td>= 2</td>
</tr>
<tr>
<td>in3</td>
<td>= 3</td>
</tr>
</tbody>
</table>

\[ \text{output}_0 = \text{"isosceles"} \]

\[ \text{intracen}_0 \]
Modelling

\[ a = \text{in1} \]
\[ b = \text{in2} \]
\[ c = \text{in3} \]

\[ \neg \text{intrace}_0 \lor AB_1 \lor a_0 = \text{in1} \]
\[ \neg \text{intrace}_0 \lor AB_2 \lor b_0 = \text{in2} \]
\[ \neg \text{intrace}_0 \lor AB_3 \lor c_0 = \text{in3} \]

\[ \neg \text{intrace}_0 \rightarrow \neg \text{intrace}_1 \]
\[ AB_4 \rightarrow \neg \text{intrace}_1 \]
\[ \neg \text{intrace}_0 \lor AB_4 \lor \neg[(a_0<0) \land (b_0>0) \land (c_0>0)] \leftrightarrow \text{intrace}_1 \]

\[ \neg \text{intrace}_1 \lor AB_5 \lor \text{output}_0 = \text{“No triangle”} \]

\[ \text{output}_0 = \text{“isosceles”} \]

\[ \text{in1} = 2 \]
\[ \text{in2} = 2 \]
\[ \text{in3} = 3 \]
Traffic Light Example

1. public static void test () {
2.     TrafficLight tl = new TrafficLight(0); // initializes tl.state=0;
3.     int i = 0;
4.     int finalState;
5.     while (i < 5) {
6.         tl.printState();
7.         if (tl.state == 0) {
8.             tl.state = 1;
9.         } else {
10.            if (tl.state == 1) {
11.                tl.state = 2;
12.            } else {
13.                if (tl.state == 2) {
14.                    tl.state = 3;
15.                } else {
16.                    tl.state = 3; // should be tl.state = 0;
17.                }
18.            }
19.        }
20.        i++;
21.    }
22.    tl.printState();
23.    finalState = tl.state;
24. }

1. Find a test case that reveals the fault
2. Build a Minion Model
3. Find all minimal diagnoses (single, double)
Outline

- Model-based debugging for Spreadsheets
  - Dependency-based models
- Model-based software debugging for Java programs
  - SSA form
  - Loop unrolling
  - Path modelling
  - Combined approaches
Combined approaches

- Refining Spectrum-based Rankings
  → **DEPUTO**

- Spectrum-based Reasoning
  → **BARINEL**

- Spectrum Enhanced Dynamic Slicing
  → **SENDYS**
STEP 1: Spectrum based Ranking

1st ranked stmt.

2nd ranked stmt.

3rd ranked stmt.

...

STEP 2: Model based Debugging (Filter)

BARINEL [2]

Spectra information of neg. TC

Hitting Sets

Diagnoses

Bayes’ rule

Ranking

\[ \Pr(d_k | obs_i) = \frac{\Pr(obs_i | d_k)}{\Pr(obs_i)} \cdot \Pr(d_k | obs_{i-1}) \]

\[ \Pr(obs_i | d_k) = \begin{cases} 
0 & \text{if } obs_i \land d_k \text{ are inconsistent}; \\
1 & \text{if } obs_i \text{ is unique to } d_k; \\
\varepsilon & \text{otherwise.} 
\end{cases} \]

\[ \varepsilon = \begin{cases} 
\prod_{j \in d_k \land a_{ij} = 1} h_j & \text{if } e_i = 0 \\
1 - \prod_{j \in d_k \land a_{ij} = 1} h_j & \text{if } e_i = 1 
\end{cases} \]

SENDYS [3]

Slices as Conflict Sets → Hitting Sets → Diagnoses → Probabilities → Diagnoses Likelihood

\[ p(\Delta_i) = \prod_{s \in \Delta_i} p_F(s) \times \prod_{s' \in \Pi \setminus \Delta_i} (1 - p_F(s')) \]

Comparison

- **DEPUTO**
  - performs best
  - but not scalable to large programs

- **BARINEL and SENDYS**
  - good alternatives for larger programs
## Summary

<table>
<thead>
<tr>
<th>Approach</th>
<th>SFL</th>
<th>MBSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>User input</td>
<td>correct / faulty</td>
<td>expected values</td>
</tr>
<tr>
<td>Fault Complexity</td>
<td>single faults</td>
<td>multiple faults</td>
</tr>
<tr>
<td>Computational Complexity</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>Granularity</td>
<td>block level</td>
<td>Stmt. level</td>
</tr>
<tr>
<td>Result</td>
<td>ranking</td>
<td>filtered set</td>
</tr>
</tbody>
</table>