Selected Topics of Software Technology 3
Model-based Software Debugging

Birgit Hofer
Institute for Software Technology
DID YOU SEE ANY ERRORS ON THE SPREADSHEET I PUT TOGETHER?

ONLY THREE.

WHAT ARE THEY?
YOUR DATA, YOUR FORMAT, AND YOUR FORMULAS.
Visualization

Design & Maintenance Support

Modeling

Static Analysis

Debugging

Testing

Fault localization

Model-based

Repair

Genetic

Spectrum-based

Spreadsheet Quality Assurance Techniques

Fault Localization
Automatic support for user to faster identify the location(s) of faults

Set of explanations
  e.g. Model-based Software Debugging (MBSD)

Ranking of Statements
  e.g. Spectrum-based Fault Localization (SFL)
Outline

- Model-based diagnosis for circuits
- Model-based debugging for Spreadsheets
  - Automatic reasoning
  - Automatic derivation of models
  - Dependency-based models
- Model-based software debugging for Java programs
Diagnosis Greek

The art of identifying a disease from its signs and symptoms
Relevant Literature


Model-based Debugging

- Origin
  - Hardware Debugging

- Basic idea
  - Derive model directly from a faulty program

- Diagnosis problem (SD, COMP, OBS)
  - System descriptions SD
  - Set of components COMP
  - Set of observations OBS

- Health predicates AB(C)
Example from the Hardware Domain

System description SD

Components COMP

Observations OBS

A true
B true
C false
D false
E false

Conflict C

AND

NOT

OR
Hitting Set Computation - Algorithm

1. Create root node for the directed graph $g$
2. Select $C$ from $CO$ with lowest cardinality
3. For each element $i$ in $C$
   a) Generate a new node $m$
   b) Draw an edge from the root to $m$, label it with $i$
4. While (CO is not empty)
   a) Select $C$ from CO with lowest cardinality
   b) For each branch $b$ in $g$ do
      o if ($b \cap C = \emptyset$) then add $C$ to $b$
5. Mark minimal hitting sets with $\sqrt{\phantom{x}}$, non-minimal hitting sets with $x$

This is a simple (but inefficient) algorithm for computing hitting sets.
Examples for Hitting Set Computations

- CO=\{\{1,2,3\},\{1,2\},\{2,3,4\}\}
  \[\Delta_1=\{2\}, \Delta_2=\{1,3\} \Delta_3=,\{1,4\}\]

- CO=\{\{1,2,3\},\{2,3\},\{1,4,5\},\{3,7\}\}
  \[\Delta_1=\{1,3\}, \Delta_2=\{3,4\} \Delta_3=,\{3,5\}, \Delta_4=\{1,2,7\}, \Delta_5=\{2,4,7\}\]
  \[\Delta_6=,\{2,5,7\}\]

- CO=\{\{1,2\},\{2,3\},\{3\},\{1,4\}\}
  \[\Delta_1=\{1,3\}, \Delta_2=\{2,3,4\}\]
Example

true
false
false
true  ☻
false
false
true
Example – Features of a circle

Version 1
1. begin
2. \( r = d \);
3. \( c = 2 \times r \times 3.14; \)
4. \( a = r \times r \times 3.14; \)
5. end

Version 2
1. begin
2. \( r = d/2; \)
3. \( c = 2 \times d \times 3.14; \)
4. \( a = r \times r \times 3.14; \)
5. end

Version 3
1. begin
2. \( r = d/2; \)
3. \( c = r \times 3.14; \)
4. \( a = r \times r \times 3.14 / 4; \)
5. end

Version 4
1. begin
2. \( r = d/2; \)
3. \( c = r \times 3.14; \)
4. \( a = r \times r \times 3.14; \)
5. end

Test case: \{d=2\}
Expected result: \{c=6.28, a=3.14\}
Outline

- Model-based diagnosis for circuits
- Model-based debugging for Spreadsheets
  - Automatic reasoning
  - Automatic derivation of models
  - Dependency-based models
- Model-based software debugging for Java programs
## Running Example

This is a simplified version of the homework/Budgetone spreadsheet from the EUSES Spreadsheet Corpus.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Item</td>
<td>1st Qtr</td>
<td>2nd Qtr</td>
<td>Total</td>
</tr>
<tr>
<td>2 Units Sold</td>
<td>1000</td>
<td>1500</td>
<td>2500</td>
</tr>
<tr>
<td>3 ASP/Unit</td>
<td>$20</td>
<td>$21</td>
<td>$20,6</td>
</tr>
<tr>
<td>4 Sales Revenue</td>
<td>$20.000</td>
<td>$31.500</td>
<td>$51.500</td>
</tr>
<tr>
<td>5 Expenses</td>
<td>$5.000</td>
<td>$6.000</td>
<td>$5.000</td>
</tr>
<tr>
<td>6 Operating Income</td>
<td>$15.000</td>
<td>$25.500</td>
<td>$46.500</td>
</tr>
<tr>
<td>7 Op Income in %</td>
<td>75,0 %</td>
<td>81,0 %</td>
<td>90,3 %</td>
</tr>
</tbody>
</table>

Should be 78.6%
## Running Example – Formula View

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Item</strong></td>
<td><strong>1st Qtr</strong></td>
<td><strong>2nd Qtr</strong></td>
<td><strong>Total</strong></td>
</tr>
<tr>
<td>2</td>
<td>Units Sold</td>
<td>1000</td>
<td>1500</td>
</tr>
<tr>
<td>3</td>
<td>ASP/Unit</td>
<td>$20</td>
<td>$21</td>
</tr>
<tr>
<td>4</td>
<td>Sales Revenue</td>
<td>$20,000</td>
<td>$31,500</td>
</tr>
<tr>
<td>5</td>
<td>Expenses</td>
<td>$5,000</td>
<td>$6,000</td>
</tr>
<tr>
<td>6</td>
<td>Operating Income</td>
<td>$15,000</td>
<td>$25,500</td>
</tr>
<tr>
<td>7</td>
<td>Op Income in %</td>
<td>75,0%</td>
<td>81,0%</td>
</tr>
</tbody>
</table>

The calculated Op Income in % should be 78.6%.
Outline

- Model-based diagnosis for circuits
- Model-based debugging for Spreadsheets
  - Automatic reasoning
    - Automatic derivation of models
    - Dependency-based models
- Model-based software debugging for Java programs
Automatic Reasoning

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

System description SD:

\[
\begin{align*}
AB(D2) \lor D2 &= B2 + C2 \\
AB(D3) \lor D3 &= D4 / D2 \\
AB(B4) \lor B4 &= B3 \times B2 \\
&\ldots
\end{align*}
\]

Test case OBS:

\[
\begin{align*}
D7 &= 0.786 \\
B7 &= 0.750 \\
&\ldots
\end{align*}
\]

For single faults:

\[
\text{SUM}(AB(D2), AB(D3), \ldots) = 1
\]

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

- Double Fault:
  - \{D3, D4\}
  - \ldots
Outline

- Model-based diagnosis for circuits
- Model-based debugging for Spreadsheets
  - Automatic reasoning
  - Automatic derivation of models
  - Dependency-based models
- Model-based software debugging for Java programs
Automatic derivation of models

**Algorithm 2** \textsc{convertSpreadsheet}

**Require:** Spreadsheet $\Pi$

**Ensure:** Set of constraints $\text{CON}_\Pi$ representing the spreadsheet $\Pi$

1: Let $\text{CON}_\Pi$ be an empty set
2: for cell $c \in \Pi$ do
3: \hspace{1em} if $c$ is a formula cell then
4: \hspace{2em} $[\text{con}_1, \text{aux}] = \text{CONVERTEXPRESSION}(\ell(c))$
5: \hspace{2em} $\text{con}_c = \text{EQUAL}(c, \text{aux})$
6: \hspace{2em} $\text{CON}_\Pi = \text{CON}_\Pi \cup \text{con}_1 \cup (AB_c \lor \text{con}_c)$
7: \hspace{1em} end if
8: end for
9: return $\text{CON}_\Pi$

All models are automatically derived from a faulty spreadsheet and also contain the fault!
Convert Expression I

- **Input:** Expression e
- **Output:** Set of Constraints and name of a variable

1: if $e$ is a cell name or constant then
2: return $[\emptyset, e]$
3: end if

4: if $e$ is of the form $(e)$ then
5: Let $[\text{CON}, aux] = \text{CONVERT EXPRESSION}(e)$
6: return $[\text{CON}, aux]$
7: end if
Convert Expression II

8: if $e$ is of the form $e_1 \circ e_2$ then
9: Let $[\text{CON}_1, aux_1] = \text{CONVERT}\text{EXPRESSION}(e_1)$
10: Let $[\text{CON}_2, aux_2] = \text{CONVERT}\text{EXPRESSION}(e_2)$
11: Generate a new variable result
12: Create a new constraint CON accordingly to the given operator $\circ$, which defines the relationship between $aux_1$, $aux_2$, and result
13: return $[\text{CON}_1 \cup \text{CON}_2 \cup \text{CON}, result]$
14: end if

Examples: A1+A2, 3 * A1, 2 * (A1 + A10)
Example: Minion Constraints Expression A1+B2-C3

MINION 3
**VARIABLES**
DISCRETE A1\{-2000..5000\}
DISCRETE B2\{-2000..5000\}
DISCRETE C2\{-2000..5000\}
DISCRETE aux1\{-2000..5000\}
DISCRETE aux2\{-2000..5000\}
**CONSTRAINTS**
sumleq([A1,B2],aux1)
sumgeq([A1,B2],aux1)
weightedsumleq([1,-1],[aux1,C2], aux2)
weightedsumgeq([1,-1],[aux1,C2], aux2)
Convert Expression III

15: if $e$ is of the form $\text{if}(e_1;e_2;e_3)$ then
16: Let $[\text{CON}_1, \text{aux}_1] = \text{CONVERT}\text{EXPRESSION}(e_1)$
17: Let $[\text{CON}_2, \text{aux}_2] = \text{CONVERT}\text{EXPRESSION}(e_2)$
18: Let $[\text{CON}_3, \text{aux}_3] = \text{CONVERT}\text{EXPRESSION}(e_3)$
19: Generate a new variable $\text{result}$
20: return $[\text{CON}_1 \cup \text{CON}_2 \cup \text{CON}_3 \cup \Psi(\text{aux}_1, \text{aux}_2, \text{aux}_3, \text{result}), \text{result}]$
21: end if
22: if $e$ is of the form $\text{sum}(c_1;c_2)$ then
23: Generate a new variable $\text{result}$
24: return $[\text{SUM}(c_1...c_2, \text{result}), \text{result}]$
25: end if
26: if $e$ is of the form $\text{and}(c_1;c_2)$ or $\text{and}(c_1, \ldots, c_2)$ then
27: Generate a new variable $\text{result}$
28: return $[\text{AND}(c_1...c_2, \text{result}), \text{result}]$
29: end if
Example: Minion Constraints for Conditional Expression \( \text{IF}(A1 > A2; A1; A2) \)

MINION 3
**VARIABLES**
BOOL aux1
BOOL aux4
BOOL aux5
DISCRETE A1\{-2000..5000\}
DISCRETE A2\{-2000..5000\}
**CONSTRAINTS**
diseq(aux1, aux4)
reify(ineq(A2, A1, -1), aux1)
reifyimply(eq(aux5, A1), aux1)
reifyimply(eq(aux5, A2), aux4)
Does the Convert Algorithm sound familiar?

It is a simple compiler, translating formulas into constraints.
Constraint-based Debugging I

- **Input:** Spreadsheet $\pi$, test case $T$
- **Output:** Set of Diagnoses $D$

1. $D \leftarrow \emptyset$
2. $\text{cones} \leftarrow \emptyset$
3. **for all** cell $c \in T$ **do**
4. \hspace{1em} $\text{cones} = \text{cones} \cup \text{CONE}(c)$
5. **end for**
6. $\text{CON}_H \leftarrow \text{CONVERTSPREADSHEET}(\text{cones})$
7. $\text{CON}_{TC} \leftarrow \text{CONVERTTEST}(T)$
8. $i \leftarrow 1$
Constraint-based Debugging II

9: while $i \leq |\Pi|$ do
10: \[ \text{CON}_{AB} = \{\text{COUNT}(AB, i)\} \]
11: \[ D \leftarrow \text{CONSTRAINTSolver}(\text{CON}_{\Pi} \cup \text{CON}_{TC} \cup \text{CON}_{AB}) \]
12: if $D \neq \emptyset$ then
13: \[ \text{return } D \]
14: else
15: \[ i \leftarrow i + 1 \]
16: end if
17: end while
18: return $D$
Example: Minion Constraints I

MINION 3
**VARIABLES**
DISCRETE B2 {-2000..5000}
...
DISCRETE D4 {-2000..5000}
BOOL ab[7]
**SEARCH**
VARORDER [ab]
PRINT ALL
Outline

- Repetition
  - Model-based diagnosis for circuits
  - Automatic derivation of models
- Model-based debugging for Spreadsheets
  - Automatic reasoning
    - Minion tutorial
  - Dependency-based models
- Model-based software debugging for Java programs
Outline

- Repetition
  - Model-based diagnosis for circuits
  - Automatic derivation of models
- Model-based debugging for Spreadsheets
  - Automatic reasoning
    - Minion tutorial
  - Dependency-based models
- Model-based software debugging for Java programs
Repetition: Hardware Debugging

System description SD

Components COMP

true
true
false
false
false
true

AND

OR

NOT

A
B
C
D
S
E

Observations OBS

true
false
false
false

• Basic idea of MBD?
• What is a conflict?
• What is a diagnosis Δ?
• What is a minimal Δ?
Example – Features of a circle

<table>
<thead>
<tr>
<th>Version 1</th>
<th>Version 2</th>
<th>Version 3</th>
<th>Version 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. begin</td>
<td>1. begin</td>
<td>1. begin</td>
<td>1. begin</td>
</tr>
<tr>
<td>2. ( r = d );</td>
<td>2. ( r = \frac{d}{2} );</td>
<td>2. ( r = \frac{d}{2} );</td>
<td>2. ( r = \frac{d}{2} );</td>
</tr>
<tr>
<td>3. ( c = 2 \times r \times 3.14 );</td>
<td>3. ( c = 2 \times \frac{d}{2} \times 3.14 );</td>
<td>3. ( c = r \times 3.14 );</td>
<td>3. ( c = r \times 3.14 );</td>
</tr>
<tr>
<td>4. ( a = r \times r \times 3.14 );</td>
<td>4. ( a = \frac{r \times r \times 3.14}{4} );</td>
<td>4. ( a = r \times r \times 3.14 );</td>
<td>4. ( a = r \times r \times 3.14 );</td>
</tr>
<tr>
<td>5.end</td>
<td>5.end</td>
<td>5.end</td>
<td>5.end</td>
</tr>
</tbody>
</table>

Test case: \{d=2\}
Expected result: \{c=6.28, a=3.14\}
Repetition: Automatic derivation of models

Compiler that translates formulas into constraints
Outline

- Repetition
  - Model-based diagnosis for circuits
  - Automatic derivation of models

- Model-based debugging for Spreadsheets
  - Automatic reasoning
    - Minion tutorial
  - Dependency-based models

- Model-based software debugging for Java programs
### Running Example

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Worker</td>
<td>Mo</td>
<td>Tu</td>
<td>We</td>
<td>Th</td>
<td>Fr</td>
<td>Total</td>
<td>Overtime</td>
<td>$/h</td>
</tr>
<tr>
<td>2</td>
<td>Jones</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>42</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>Smith</td>
<td>9</td>
<td>9</td>
<td>10</td>
<td>7</td>
<td>8</td>
<td>43</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>85</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

5

6 Regular working time 40

7 Working hours Jones 42

8 Working hours Smith 43

"My wage is too low!"
# VARIABLES

BOOL AB_G4, AB_H4, AB_J2, AB_J3, AB_G2, AB_G3  // abnormal variables
BOOL AB_H2, AB_H3, AB_E7, AB_E8

INT G4, H4, J2, J3  // Output cells
INT G2, G3, H2, H3, E7, E8  // Intermediate cells

BOOL aux_cond, aux_neg  // Auxiliary variables for H2
INT aux_then, aux_else

BOOL aux1_cond, aux1_neg  // Auxiliary variables for H3
INT aux1_then, aux1_else

INT aux2, aux3, aux4, aux5  // Other auxiliary variables
Running Example – Pseudo Code II

# TEST CASE
# INPUT
eq(B2,9) eq(D3,10) eq(F3,8)
eq(B3,9) eq(E2,8) eq(I2,20)
eq(C2,9) eq(E3,7) eq(I3,24)
eq(C3,9) eq(F2,8) eq(E6,40)
eq(D2,8)

# EXPECTED OUTPUT
eq(G4,85) // correct value
eq(J2,860) // correct value
eq(H4,5) // computed value=3, desired value=5
greater(J3,1044) // Smith's wage should be larger than 1044
Running Example – Pseudo Code III

# FORMULAS

watched-or(\(AB\_G2=\text{true}\), \(B2 + C2 + D2 + E2 + F2 = G2\))

watched-or(\(AB\_G3=\text{true}\), \(B3 + C3 + D3 + E3 + F3 = G3\))

watched-or(\(AB\_G4=\text{true}\), \(G2 + G3 = G4\))

watched-or(\(AB\_H4=\text{true}\), \(H2 + H3 = H4\))

watched-or(\(AB\_E7=\text{true}\), eq(G2,E7))

NOTES

The constraint watched-or(\(\{C1,...,Cn\}\)) ensures that at least one of the constraints \(C1,...,Cn\) is true.
# FORMULAS

AB_H2 OR (H2 = IF(G2>E6;G2-E6;0))

- watched-or(AB_H2=true, reify(greater(G2,E6,aux_cond)))  // condition
- watched-or(AB_H2=true, diseq(aux_cond,aux_neg))        // negation of the condition
- watched-or(AB_H2=true, G2 - E6 = aux_then)              // then_value
- watched-or(AB_H2=true, eq(aux_else,0))                  // else_value
- watched-or(AB_H2=true, reifyimply(eq(H2, aux_then),aux_cond)) // taking value of then-branch
- watched-or(AB_H2=true, reifyimply(eq(H2, aux_else),aux_neg)) // taking value of else-branch

NOTES

- **reify(constraint, r)** ensures that the Boolean variable r is set to 1 iff constraint is satisfied. If r is 0 the constraint must NOT be satisfied; if r is 1 it must be satisfied. If the constraint is satisfied then r must be 1, and if not then r must be 0.

- **reifyimply(constraint, r)** only checks that if r is set to 1 then constraint must be satisfied. If r is not 1, constraint may be either satisfied or unsatisfied. Furthermore r is never set by propagation, only by search; Satisfaction of constraint does not affect the value of r.
Running Example – Pseudo Code V

### FORMULAS

#### AB_H3 OR (H3 = IF(G3>E6;G3-E7;0))

watched-or(AB_H3=true, reify(greater(G3,E6,aux1_cond)))  // condition

watched-or(AB_H3=true, diseq(aux1_cond,aux1_neg))  // negation of the condition

watched-or(AB_H3=true, G3 - E7 = aux1_then)  // then_value

watched-or(AB_H3=true, eq(aux1_else,0))  // else_value

watched-or(AB_H3=true, reifyimply(eq(H3, aux1_then),aux1_cond))  // taking value of then-branch

watched-or(AB_H3=true, reifyimply(eq(H3, aux1_else),aux1_neg))  // taking value of else-branch

#### AB_J2 OR (J2 = I2*(G2+H2*0.5))

watched-or(AB_J2=true, H2 * 0.5 = aux2)

watched-or(AB_J2=true, G2 + aux2 = aux3)

watched-or(AB_J2=true, I2 * aux3 = J2)

#### AB_J3 OR (J3 = I3*(G3+H3*0.5))

watched-or(AB_J3=true, H3 * 0.5 = aux4)

watched-or(AB_J3=true, G3 + aux4 = aux5)

watched-or(AB_J3=true, I3 * aux5 = J3)

### RESTRICT DIAGNOSIS SIZE

SUM([AB_G4, AB_H4, AB_J2, AB_J3, AB_G2, AB_G3, AB_H2, AB_H3, AB_E7, AB_E8],1)
Running Example – Questions

1. Why do we need the watched-or?

2. For cell H2, we use the reify for modeling the condition. Why?

3. What cells should be reported as single fault diagnoses?

4. How can we compute the double fault diagnoses?

5. What cells should be reported as double fault diagnoses?
Minion

A very short tutorial
File Structure

MINION 3
**VARIABLES**
...
**CONSTRAINTS**
...
**SEARCH**
VARORDER [var1,var2,...]
PRINT [var1,var2,...] // alternative PRINT ALL, PRINT NONE
**EOF**
Minion variable types

- **Boolean**
  
  `BOOL bo`

- **Discrete**
  
  `DISCRETE d {1..3}`

- **Bound**
  
  `BOUND b {1..3}`

- **Sparsebound**
  
  `SPARSEBOUND s {1,3,6,7}`

- **Matrices of variables**
  
  `DISCRETE q[3] {0..5} // array`
  
  `BOOL bm[2,2] // 2x2 matrix`

---

Basic rule of thumb

Always use `BOOL` for Boolean domains, `DISCRETE` for domains of size up to around 100, and the `BOUND`. Use `SPARSEBOUND` over `BOUND` if the domain is sparse, e.g. `{1; 10; 100; 1000}`.
Important constraints I

- **Equality**
  \[ \text{eq}(\text{var1}, \text{var2}) \]

- **Unequality**
  \[ \text{diseq}(v0, v1) \]

- **Comparison**
  \[ \text{ineq}(x, y, k) \]  # ensures that \( x \leq y + k \)
  \[ \text{ineq}(x, y, -1) \]  # ensures that \( x < y \)
Important constraints II

- **Sum**
  
  \[
  \text{sumgeq}(\text{vec}, c) \quad \# \text{ ensures that } \text{sum}(\text{vec}) \geq c \\
  \text{sumleq}(\text{vec}, c) \quad \# \text{ ensures that } \text{sum}(\text{vec}) \leq c
  \]

  Example: B2+C2=D2
  
  \[
  \text{sumgeq}([B2,C2],D2) \\
  \text{sumleq}([B2,C2],D2)
  \]

- **Multiplication**
  
  \[
  \text{product}(x,y,z) \quad \# \text{ ensures that } z=xy
  \]
Important constraints III

- **Minus**
  \[
  \text{weightedsumgeq(constVec, varVec, total)} \quad \# \text{ ensures that constVec.varVec(scalar dot product) } \geq \text{ total}
  \]
  \[
  \text{weightedsumleq(constVec, varVec, total)} \quad \# \text{ ensures that constVec.varVec(scalar dot product) } \leq \text{ total}
  \]

  Example: B4-B5=B6
  \[
  \text{weightedsumgeq([1,-1],[B4,B5],B6)}
  \]
  \[
  \text{weightedsumleq([1,-1],[B4,B5],B6)}
  \]

- **Division**
  \[
  \text{div}(x,y,z) \quad \# \text{ ensures floor}(x/y)=z, \text{ e.g. } 10/3 = 3; -10/3 = -4
  \]
Important constraints IV

- Reification (for Modeling Conditionals)

  \texttt{reify(constraint, r)} \quad # \text{ensures that the Boolean variable } r \text{ is set to } 1 \text{ iff constraint is satisfied. If } r \text{ is } 0 \text{ the constraint must NOT be satisfied; if } r \text{ is } 1 \text{ it must be satisfied. If the constraint is satisfied then } r \text{ must be } 1, \text{ and if not then } r \text{ must be } 0.

  \texttt{reifyimply(constraint, r)} \quad # \text{only checks that if } r \text{ is set to } 1 \text{ then constraint must be satisfied. If } r \text{ is not } 1, \text{ constraint may be either satisfied or unsatisfied. Furthermore } r \text{ is never set by propagation, only by search; Satisfaction of constraint does not affect the value of } r.
Important constraints V

- AB or Formula
  \[\text{watched-or}\{C_1, \ldots, C_n\}\] # ensures that at least one of the constraints \(C_1, \ldots, C_n\) is true

- Report small diagnoses
  \[\text{watchsumgeq}(\text{vec}, c)\] # ensures that \(\text{sum}(\text{vec}) \geq c\)
  \[\text{watchsumleq}(\text{vec}, c)\] # ensures that \(\text{sum}(\text{vec}) \leq c\)

Example: Single fault diagnoses
\[\text{watchsumgeq}([\text{AB\_G4, AB\_H4, AB\_J2, AB\_J3}], 1)\]
\[\text{watchsumleq}([\text{AB\_G4, AB\_H4, AB\_J2, AB\_J3}], 1)\]
Solver call

- minion -findallsols fileName.minion

- See documentation for configuration parameters
## Example II

<table>
<thead>
<tr>
<th>A</th>
<th>1st Qtr</th>
<th>2nd Qtr</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Units Sold</td>
<td>100</td>
<td>100</td>
<td>200</td>
</tr>
<tr>
<td>ASP/Unit</td>
<td>$20</td>
<td>$30</td>
<td>$25,0</td>
</tr>
<tr>
<td>Sales Revenue</td>
<td>$2,000</td>
<td>$3,000</td>
<td>$5,000</td>
</tr>
<tr>
<td>Expenses</td>
<td>$500</td>
<td>$300</td>
<td>$300</td>
</tr>
<tr>
<td>Operating Income</td>
<td>$1,500</td>
<td>$2,700</td>
<td>$4,700</td>
</tr>
<tr>
<td>Op Income in %-Sales</td>
<td>75.0%</td>
<td>90.0%</td>
<td>94.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>1st Qtr</th>
<th>2nd Qtr</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Units Sold</td>
<td>100</td>
<td>100</td>
<td>=SUM(B2:C2)</td>
</tr>
<tr>
<td>ASP/Unit</td>
<td>20</td>
<td>30</td>
<td>=D4/D2</td>
</tr>
<tr>
<td>Sales Revenue</td>
<td>=B3*B2</td>
<td>=C3*C2</td>
<td>=SUM(B4:C4)</td>
</tr>
<tr>
<td>Expenses</td>
<td>500</td>
<td>300</td>
<td>=SUM(C5:C5)</td>
</tr>
<tr>
<td>Operating Income</td>
<td>=B4-B5</td>
<td>=C4-C5</td>
<td>=D4-D5</td>
</tr>
<tr>
<td>Op Income in %-Sales</td>
<td>=B6/B4</td>
<td>=C6/C4</td>
<td>=D6/D4</td>
</tr>
</tbody>
</table>
Example II - Tasks

- Write the Minion model for this spreadsheet

- Compute all
  - Single fault diagnoses
  - Double fault diagnoses
Outline

- Repetition
  - Model-based diagnosis for circuits
  - Automatic derivation of models
- Model-based debugging for Spreadsheets
  - Automatic reasoning
  - Minion tutorial
  - Dependency-based models
- Model-based software debugging for Java programs
Automatic Reasoning

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

System description SD:

\[ \text{AB(D2)} \lor D2 = B2 + C2 \]
\[ \text{AB(D3)} \lor D3 = D4 / D2 \]
\[ \text{AB(B4)} \lor B4 = B3 \times B2 \]
...

Test case OBS:

\[ D7 = 0.786 \]
\[ B7 = 0.750 \]
...

For single faults:

\[ \text{SUM(AB(D2), AB(D3), \ldots)} = 1 \]

Should be 78.6%

For single faults:

\[ \text{SUM(AB(D2), AB(D3), \ldots)} = 1 \]

\[ \text{AB(D2)} \lor \text{behavior(D2)} \]
\[ \text{AB(D3)} \lor \text{behavior(D3)} \]
\[ \text{AB(B4)} \lor \text{behavior(B4)} \]

Single Fault:

- \{D5\}
- \{D6\}
- \{D7\}

Double Fault:

- \{D3, D4\}
...

More General:

\[ \text{AB(D2)} \lor \text{behavior(D2)} \]
\[ \text{AB(D3)} \lor \text{behavior(D3)} \]
\[ \text{AB(B4)} \lor \text{behavior(B4)} \]
Models for a Spreadsheet’s Behavior

Value-based
- \( D2 = B2 + C2 \)
- \( D3 = D4 / D2 \)
- \( B4 = B3 \times B2 \)

Dependency-based
- \( \text{ok}(B2) \land \text{ok}(C2) \rightarrow \text{ok}(D2) \)
- \( \text{ok}(D4) \land \text{ok}(D2) \rightarrow \text{ok}(D3) \)
- \( \text{ok}(B3) \land \text{ok}(B2) \rightarrow \text{ok}(B4) \)
Models for a Spreadsheet’s Behavior

Value-based

- $D_2 == B_2 + C_2$
- $D_3 == D_4 / D_2$
- $B_4 == B_3 \times B_2$

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

Dependency-based

- $ok(B_2) \land ok(C_2) \rightarrow ok(D_2)$
- $ok(D_4) \land ok(D_2) \rightarrow ok(D_3)$
- $ok(B_3) \land ok(B_2) \rightarrow ok(B_4)$

+ fast
+ only Boolean
- many diagnoses
Example II – Continuation

- Replace all discrete variables with Boolean variables

- Replace the test case 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 \( \min([\text{var}1,\text{var}2,...],\text{result}) \) to model the 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 cause for the difference!

- Do you have any ideas how to improve the dependency-based model?
Improving the Dependency-based Model

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

- 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

<table>
<thead>
<tr>
<th>Item</th>
<th>1st Qtr</th>
<th>2nd Qtr</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units Sold</td>
<td>1000</td>
<td>1500</td>
<td>SUM(B2:C2)</td>
</tr>
<tr>
<td>ASP/Unit</td>
<td>20</td>
<td>21</td>
<td>D4/D2</td>
</tr>
<tr>
<td>Sales Revenue</td>
<td>=B3*B2</td>
<td>=C3*C2</td>
<td>SUM(B4:C4)</td>
</tr>
<tr>
<td>Expenses</td>
<td>5000</td>
<td>6000</td>
<td>SUM(B5:B5)</td>
</tr>
<tr>
<td>Operating Income</td>
<td>=B4-B5</td>
<td>=C4-C5</td>
<td>D4-D5</td>
</tr>
<tr>
<td>Op Income in %</td>
<td>=B6/B4</td>
<td>=C6/C4</td>
<td>D6/D4</td>
</tr>
</tbody>
</table>
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

63 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 (-16.6%)</td>
<td>38.6 (-14.2%)</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>
Summary of different models

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

- **Value-based (VB)**
  \[ D_2 = B_2 + C_2 \]
  + 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
  + low computation time
    - less than 1 second

- **Improved dep.-based (IDB)**
  ok(B2) ∧ ok(C2) ↔ ok(D2)
  ~ reduced number of diagnoses
  + low computation time
    - less than 1 second

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

**3.3 times more than VB**
**15% less diagnoses than DB**