Automated debugging – the past, the now, and the future

*Part 2: Debugging based on models*

Franz Wotawa
TU Graz, Institute for Software Technology
wotawa@ist.tugraz.at
Content

• The debugging problem
• Conversion of programs into constraints
• Specification knowledge / handling functions
• Testing

• Joint work with: Bernhard Aichernig, R. Ceballos, Gerhard Friedrich, Wolfgang Maier, Julia Nica, Mihai Nica, Simona Nica, Ingo Pill, Markus Stumptner, Jörg Weber, Dominik Wieland.
The debugging problem

• **Given:**
  – Source code of a program
  – A test suite comprising at least one failing test case

• **Wanted:**
  – Root cause for the detected misbehavior (statement, expression,..)
Debugging – A (very) short intro

1. begin
2. \(i = 2 \times x;\)
3. \(j = 2 \times y;\)
4. \(o1 = i + j;\)
5. \(o2 = i \times i;\)
6. end;

Debugger

Diagnoses?

\(x = 1, y = 2, o1 = 8, o2 = 4\)
Debugging using constraints

1. begin
2. \( i = 2 \times x; \)
3. \( j = 2 \times y; \)
4. \( o_1 = i + j; \)
5. \( o_2 = i \times i; \)
6. end;

\( x = 1, \ y = 2, \ o_1 = 8, \ o_2 = 4 \)

Programm execution

Ab(2) \lor \ i = 2 \times x;
Ab(3) \lor \ j = 2 \times y;
Ab(4) \lor \ o_1 = i + j;
Ab(5) \lor \ o_2 = i \times i;

\( x = 1 \)
\( y = 2 \)
\( o_1 = 8 \)
\( o_2 = 4 \)

Constraint solving / equation solving
Finding bugs using constraints

\[ \neg \text{Ab}(2) \land \neg \text{Ab}(3) \land \neg \text{Ab}(4) \land \neg \text{Ab}(5) \]

\[ j = 2 \times 2 = 4 \]
\[ o1 = i + j = 8 = i + 4 \rightarrow i = 4 \]
\[ o2 = 4 = i \times i = 4 \times 4 \rightarrow \text{FAIL!!!!} \]

\[ \neg \text{Ab}(2) \land \text{Ab}(3) \land \neg \text{Ab}(4) \land \neg \text{Ab}(5) \]

\[ i = 2 \times 1 = 2 \]
\[ o1 = 8 = 2 + j \rightarrow j = 6 \]
\[ o2 = 4 = i \times i = 2 \times 2 \]

And so on ... finally leading to 2 possible diagnoses statement 3 and statement 4
Observations

• Reasoning in „all directions“ (from input to outputs and vice versa)

• Make assumptions about the correctness of „components“

• Use in-consistencies for accepting or refuting assumptions
Additional remarks

We might try to find root causes by tracing back dependencies and eliminating candidates that also contribute to correct output values.

This should NOT be done because of failure masking:
Basic definitions

Definition 1 (Diagnosis System) A diagnosis system \((SD, COMP)\) consists out of a system description \(SD\), i.e., a set of FOL sentences describing the components behavior and the systems structure, and a set of diagnosis components \(COMP\).

\[ \neg AB(M_1) \rightarrow x = a \ast c \quad (or \ AB(M_1) \lor x = a \ast c) \]

\[ \neg AB(M_2) \rightarrow y = b \ast d \]

....

AB...Abnormal / Assumption
Definition 2 (Diagnosis)  Let \((SD, COMP)\) be a diagnosis system and \(OBS\) a set of observations. A set \(\Delta \subseteq COMP\) is a diagnosis for the diagnosis problem \((SD, COMP, OBS)\) iff

\[
SD \cup OBS \cup \{\neg ab(C) \mid C \in COMP \setminus \Delta\} \cup \{ab(C) \mid C \in \Delta\}
\]

is consistent.
What is needed?

- Mapping of programs to a model!

```
1. begin
2. i = 2 * x;
3. j = 2 * y;
4. o1 = i + j;
5. o2 = i * i;
6. end;
```

```
Ab(2) ∨ i = 2 * x;
Ab(3) ∨ j = 2 * y;
Ab(4) ∨ o1 = i + j;
Ab(5) ∨ o2 = i * i;
```
CONVERTING PROGRAMS TO CONSTRAINTS
Assumptions

• Sequential programming language without OO constructs
• The program terminates
• No exception handling
Challenges

• Loops / recursive function calls
• Variables defined more than once in a program

```c
int power(int a, int exp)
1. int e = exp;
2. int res = 1;
3. while (e > 0) {
4.    res = res * a;
5.    e = e - 1;
} 
6. return res;
```
Handling loops

• Execution of

    while (e > 0) { ... }

leads to:

    if (e > 0) { ...
        if (e > 0) { ...
            if (e > 0) { ...
                if (e > 0) { ... }}}}
Loop unrolling
Summary loop unrolling

• No influence on semantics if nesting depth set appropriately
  – Nesting depth > maximum number of iterations caused by a test case

• Increase in size of the program (accordingly to the complexity of the program)
int power_loopfree(int a, int exp)
1. int e = exp;
2. int res = 1;
3. if (e > 0) {
4.    res = res * a;
5.    e = e - 1;
6.    if (e > 0) {
7.       res = res * a;
8.       e = e - 1; } } }
9. return res;
Static single assignment form (SSA form)

- In order to convert programs to constraints every variable is only allowed to be defined once!

- **Solution**: convert the loop-free program into its SSA form
SSA form

• **Property**: No two left-side (=defined) variables have the same name

• Assign each defined variable an unique index.
• If a variable is used afterwards in the program, refer to the last given index.
Conditional statements

• Statement of the form

\[
\text{if } C \text{ then } B_1 \text{ else } B_2 \text{ end if;}
\]

• Convert \( B_1 \) and \( B_2 \) separately using a distinguished set of indices
Conditional statements

• Introduce a new function $\Phi$.
• Add a new statement

$$x_C = C;$$

• For each defined variable $x$ in either $B_1$ or $B_2$ add the following assignment:

$$x_i = \Phi(x_{\text{index}}(B_1), x_{\text{index}}(B_2), x_C);$$
Semantics of $\Phi$

$$
\Phi(v_j, v_k, \text{cond}_i) \overset{\text{def}}{=} \begin{cases} 
  v_j & \text{if } \text{cond}_i = \text{true} \\
  v_k & \text{otherwise}
\end{cases}
$$
Example (cont.)

int power_SSA(int a, int exp) {
1. int e_0 = exp;
2. int res_0 = 1;
3. bool cond_0 = (e_0 > 0);
4. int res_1 = res_0 * a;
5. int e_1 = e_0 - 1;
6. bool cond_1 = cond_0 \land (e_1 > 0);
7. int res_2 = res_1 * a;
8. int e_2 = e_1 - 1;
9. int res_3 = \Phi(res_2, res_1, cond_1);
10. int e_3 = \Phi(e_2, e_1, cond_1);
11. int res_4 = \Phi(res_3, res_0, cond_0);
12. int e_4 = \Phi(e_3, e_0, cond_0); }
Summary SSA conversion

• Only assignment statements!
• Direct conversion to constraints possible
• The conditions used in the $\Phi$ function are equivalent to the path conditions
• No substantial increase of size
Conversion to CSPs

- Conversion only needed for assignments

<table>
<thead>
<tr>
<th>SSA Statement</th>
<th>MINION Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>e_0 = exp;</td>
<td>auxVar = ComputeExpression(exp), eq(e_0, auxVar)</td>
</tr>
<tr>
<td>cond_0 = (e_0 &gt; 0);</td>
<td>reify(ineq(0,e_0,-1 ),cond_0)</td>
</tr>
<tr>
<td>cond_1 = cond_0 ∧ (e_1 &gt; 0);</td>
<td>reify(ineq(0,e_1,-1 ),cond_aux) reify(watchsumgeq([cond_0,cond_aux], 2),cond_1)</td>
</tr>
<tr>
<td>res_4 = Φ(res_3, res_0, cond_0);</td>
<td>watched-or(eq(cond_0,0), eq(res_4,res_3)) watched-or(eq(cond_0,1), eq(res_4,res_0))</td>
</tr>
</tbody>
</table>
ComputeExpression

• **Input**: An expression $E_{expr}$ and an empty set $M$ for storing the MINION constraints.

• **Output**: A set of minion constraints representing the expression stored in $M$, and a variable or constant where the result of the conversion is finally stored.
ComputeExpression (cont.)

1. If $E_{expr}$ is a variable or constant, then return $E_{expr}$.

2. Otherwise, $E_{expr}$ is of the form $E_{1_{expr}}{op}E_{2_{expr}}$.
   a) Let $aux_1 = \text{ComputeExpression}(E_{1_{expr}})$
   b) Let $aux_2 = \text{ComputeExpression}(E_{2_{expr}})$
   c) Generate a new MINON variable $result$ and create MINON constraints accordingly to the given operator $op$, which defines the relationship between $aux_1$, $aux_2$, and $result$, and add them to $M$.

3. Return $result$. 

14.06.17
HSST 2017 Halmstad, Sweden
Example

• **Example:** Given expression

\[ a_0 + b_0 - c_0 \]

• **Minion constraints:**

\[
\begin{align*}
\text{sumleq}([a_0, b_0], \text{aux1}) \\
\text{sumgeq}([a_0, b_0], \text{aux1}) \\
\text{weightedsumleq}([1, -1], [\text{aux1}, c_0], \text{aux2}) \\
\text{weightedsumgeq}([1, -1], [\text{aux1}, c_0], \text{aux2})
\end{align*}
\]
Summary conversion process

• Conversion in 3 steps:
  1. Convert program to loop-free variant (loop unrolling)
  2. Convert loop-free variant to SSA form
  3. Convert SSA form to constraint system (Minion)
Debugging / Testing

USING CONSTRAINTS FOR DEBUGGING
Input to the debugging problem

• A debugging problem comprises
  – A program, and
    
    - A failing test case

\[
\begin{align*}
1. & \quad i = 2 \times x; \\
2. & \quad j = 2 \times y; \\
3. & \quad o1 = i + j; \\
4. & \quad o2 = i \times i;
\end{align*}
\]

\[x = 1, \ y = 2, \ o1 = 8, \ o2 = 4;\]
Introduce new variable $Ab$

• Use variable to state whether a statement is assumed to work correctly or not!

1. $Ab_1 \lor i = 2 \times x$
2. $Ab_2 \lor j = 2 \times y$
3. $Ab_3 \lor o1 = i + j$
4. $Ab_4 \lor o2 = i \times i$
Debugging = CSP solving

1. Ab₁ ∨ i₁ = 2 * x;
2. Ab₂ ∨ j₁ = 2 * y;
3. Ab₃ ∨ o₁₁ = i₁ + j₁;
4. Ab₄ ∨ o₂₁ = i₁ * i₁;

x = 1, y = 2, o₁₁ = 8, o₂₁ = 4;

Convert to constraints

A solution to the resulting CSP is a solution to the debugging problem!
MINION 3

++VARIABLES++
DISCRETE x (-250..250)
DISCRETE y (-250..250)
DISCRETE I (-250..250)
DISCRETE J (-250..250)
DISCRETE O1 (-250..250)
DISCRETE O2 (-250..250)

DISCRETE I_1 (-250..250)
DISCRETE J_1 (-250..250)
DISCRETE O1_1 (-250..250)
DISCRETE O2_1 (-250..250)

BOOL ab[4]

++SEARCH++
PRINT [ [ab] ]
VARORDER [ab]

++CONSTRAINTS++

# System description
product(2, x, I_1)
product(2, y, J_1)
sumeq([1, 2], O1_1)
sumeq([1, 2], O1_1)
product([1, 2, 3])
Diagnosis algorithm

Algorithm 1 ConDiag((VAR_S, DOM, CONS ∪ COBS), COMP, n)

Input: A constraint model (VAR_S, DOM, CONS ∪ COBS) of a system having components COMP and the desired diagnosis cardinality n
Output: All minimal diagnoses up to the predefined cardinality n

1: Let DS be {}
2: Let M be CONS ∪ COBS
3: for i = 0 to n do
4: \[ CM = M \cup \{ |\{ab_C| C \in COMP \land ab_C = T\}| = i \} \]
5: \[ S = \mathcal{P}(\text{CSolver}(VAR_S, DOM, CM)) \]
6: if i is 0 and S is \{\{\}\} then
7: \hspace{1cm} return S
8: end if
9: Let DS be DS ∪ S.
10: \[ M = M \cup \{\neg (C(S))\} \]
11: end for
12: return DS

Some remarks

• Focus on small solutions (single faults)
  – Use constraint solver that searches for solutions where only on Ab variable is true!

• There must be a mapping back from the Ab variables to the statements of the original program
Results obtained

• Java implementation of the conversion process
• Use Minion Vo.8 as constraint solver
• Intel Pentium Dual Core 2 GHz with 4 GB of RAM.
• AIM is a model-based debugging tool based on abstract interpretation (from Wolfgang Mayer, Markus Stumptner)
<table>
<thead>
<tr>
<th>Variant</th>
<th>LOCπ</th>
<th>#D</th>
<th>Time(s)</th>
<th>DAIM min</th>
<th>DAIM max</th>
<th>Time(s)</th>
<th>AIM best</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcas_v01</td>
<td>78</td>
<td>28</td>
<td>0,28</td>
<td>21</td>
<td>23</td>
<td>16</td>
<td>83</td>
</tr>
<tr>
<td>tcas_v02</td>
<td>78</td>
<td>26</td>
<td>0,28</td>
<td>12</td>
<td>22</td>
<td>11</td>
<td>33</td>
</tr>
<tr>
<td>tcas_v03</td>
<td>78</td>
<td>29</td>
<td>0,32</td>
<td>2</td>
<td>23</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>tcas_v04</td>
<td>78</td>
<td>25</td>
<td>0,26</td>
<td>20</td>
<td>23</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>tcas_v05</td>
<td>78</td>
<td>25</td>
<td>0,33</td>
<td>18</td>
<td>21</td>
<td>12</td>
<td>25</td>
</tr>
<tr>
<td>tcas_v06</td>
<td>78</td>
<td>25</td>
<td>0,28</td>
<td>19</td>
<td>22</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>tcas_v07</td>
<td>78</td>
<td>9</td>
<td>0,26</td>
<td>10</td>
<td>22</td>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>tcas_v08</td>
<td>78</td>
<td>27</td>
<td>0,36</td>
<td>22</td>
<td>22</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>tcas_v09</td>
<td>78</td>
<td>11</td>
<td>0,26</td>
<td>11</td>
<td>12</td>
<td>11</td>
<td>22</td>
</tr>
<tr>
<td>tcas_v10</td>
<td>78</td>
<td>29</td>
<td>0,23</td>
<td>21</td>
<td>26</td>
<td>16</td>
<td>31</td>
</tr>
<tr>
<td>tcas_v11</td>
<td>78</td>
<td>23</td>
<td>0,31</td>
<td>17</td>
<td>24</td>
<td>12</td>
<td>29</td>
</tr>
<tr>
<td>tcas_v12</td>
<td>78</td>
<td>23</td>
<td>0,21</td>
<td>17</td>
<td>23</td>
<td>12</td>
<td>37</td>
</tr>
<tr>
<td>tcas_v13</td>
<td>78</td>
<td>27</td>
<td>0,26</td>
<td>21</td>
<td>22</td>
<td>24</td>
<td>28</td>
</tr>
<tr>
<td>tcas_v14</td>
<td>78</td>
<td>6</td>
<td>0,15</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>35</td>
</tr>
<tr>
<td>tcas_v15</td>
<td>78</td>
<td>24</td>
<td>0,25</td>
<td>18</td>
<td>21</td>
<td>13</td>
<td>19</td>
</tr>
<tr>
<td>tcas_v16</td>
<td>78</td>
<td>26</td>
<td>0,29</td>
<td>20</td>
<td>22</td>
<td>17</td>
<td>47</td>
</tr>
<tr>
<td>tcas_v17</td>
<td>78</td>
<td>9</td>
<td>0,21</td>
<td>10</td>
<td>22</td>
<td>16</td>
<td>44</td>
</tr>
<tr>
<td>tcas_v18</td>
<td>78</td>
<td>9</td>
<td>0,24</td>
<td>10</td>
<td>22</td>
<td>13</td>
<td>51</td>
</tr>
<tr>
<td>tcas_v19</td>
<td>78</td>
<td>9</td>
<td>0,26</td>
<td>10</td>
<td>22</td>
<td>14</td>
<td>24</td>
</tr>
<tr>
<td>tcas_v20</td>
<td>78</td>
<td>27</td>
<td>0,28</td>
<td>22</td>
<td>23</td>
<td>15</td>
<td>29</td>
</tr>
<tr>
<td>tcas_v21</td>
<td>78</td>
<td>27</td>
<td>0,24</td>
<td>22</td>
<td>22</td>
<td>15</td>
<td>29</td>
</tr>
<tr>
<td>Name</td>
<td>LOCprog</td>
<td>#It</td>
<td>LOCssa</td>
<td>#D</td>
<td>T_s</td>
<td>CO</td>
<td>#VarCO</td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td>-----</td>
<td>--------</td>
<td>----</td>
<td>-----</td>
<td>----</td>
<td>--------</td>
</tr>
<tr>
<td>Division_V0</td>
<td>21</td>
<td>1</td>
<td>26</td>
<td>4</td>
<td>0,01</td>
<td>24/22</td>
<td></td>
</tr>
<tr>
<td>Division_V1</td>
<td>21</td>
<td>1</td>
<td>26</td>
<td>3</td>
<td>0,01</td>
<td>24/22</td>
<td></td>
</tr>
<tr>
<td>Division_V2</td>
<td>21</td>
<td>1</td>
<td>26</td>
<td>2</td>
<td>0,01</td>
<td>24/22</td>
<td></td>
</tr>
<tr>
<td>Division_V3</td>
<td>21</td>
<td>2</td>
<td>32</td>
<td>5</td>
<td>0,01</td>
<td>33/28</td>
<td></td>
</tr>
<tr>
<td>Division_V4</td>
<td>21</td>
<td>2</td>
<td>32</td>
<td>5</td>
<td>0,01</td>
<td>33/28</td>
<td></td>
</tr>
<tr>
<td>Division_V5</td>
<td>21</td>
<td>2</td>
<td>32</td>
<td>2</td>
<td>0,01</td>
<td>33/28</td>
<td></td>
</tr>
<tr>
<td>Mult_V0</td>
<td>12</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>0,01</td>
<td>13/12</td>
<td></td>
</tr>
<tr>
<td>Mult_V1</td>
<td>12</td>
<td>1</td>
<td>20</td>
<td>4</td>
<td>0,01</td>
<td>13/12</td>
<td></td>
</tr>
<tr>
<td>Mult_V2</td>
<td>12</td>
<td>2</td>
<td>20</td>
<td>2</td>
<td>0,01</td>
<td>13/12</td>
<td></td>
</tr>
<tr>
<td>Mult_V3</td>
<td>12</td>
<td>2</td>
<td>25</td>
<td>5</td>
<td>0,01</td>
<td>21/17</td>
<td></td>
</tr>
<tr>
<td>Mult_V4</td>
<td>12</td>
<td>2</td>
<td>25</td>
<td>5</td>
<td>0,01</td>
<td>21/17</td>
<td></td>
</tr>
<tr>
<td>Mult_V5</td>
<td>12</td>
<td>2</td>
<td>25</td>
<td>2</td>
<td>0,01</td>
<td>21/17</td>
<td></td>
</tr>
<tr>
<td>MultV2.V0</td>
<td>18</td>
<td>1</td>
<td>27</td>
<td>6</td>
<td>0,01</td>
<td>24/20</td>
<td></td>
</tr>
<tr>
<td>MultV2.V1</td>
<td>18</td>
<td>1</td>
<td>27</td>
<td>6</td>
<td>0,01</td>
<td>24/20</td>
<td></td>
</tr>
<tr>
<td>MultV2.V2</td>
<td>18</td>
<td>1</td>
<td>27</td>
<td>6</td>
<td>0,01</td>
<td>24/20</td>
<td></td>
</tr>
<tr>
<td>MultV2.V3</td>
<td>18</td>
<td>2</td>
<td>48</td>
<td>6</td>
<td>0,01</td>
<td>65/49</td>
<td></td>
</tr>
<tr>
<td>MultV2.V4</td>
<td>18</td>
<td>2</td>
<td>48</td>
<td>5</td>
<td>0,01</td>
<td>65/49</td>
<td></td>
</tr>
<tr>
<td>MultV2.V5</td>
<td>18</td>
<td>2</td>
<td>48</td>
<td>8</td>
<td>0,01</td>
<td>65/49</td>
<td></td>
</tr>
<tr>
<td>Sum_V0</td>
<td>13</td>
<td>1</td>
<td>21</td>
<td>4</td>
<td>0,01</td>
<td>13/10</td>
<td></td>
</tr>
<tr>
<td>Sum_V1</td>
<td>13</td>
<td>1</td>
<td>21</td>
<td>2</td>
<td>0,01</td>
<td>13/10</td>
<td></td>
</tr>
<tr>
<td>Sum_V2</td>
<td>13</td>
<td>1</td>
<td>21</td>
<td>2</td>
<td>0,01</td>
<td>13/10</td>
<td></td>
</tr>
<tr>
<td>Sum_V3</td>
<td>13</td>
<td>2</td>
<td>26</td>
<td>5</td>
<td>0,01</td>
<td>22/16</td>
<td></td>
</tr>
<tr>
<td>Sum_V4</td>
<td>13</td>
<td>2</td>
<td>26</td>
<td>2</td>
<td>0,01</td>
<td>22/16</td>
<td></td>
</tr>
<tr>
<td>Sum_V5</td>
<td>13</td>
<td>2</td>
<td>26</td>
<td>5</td>
<td>0,01</td>
<td>22/16</td>
<td></td>
</tr>
<tr>
<td>gCD_V0</td>
<td>24</td>
<td>2</td>
<td>37</td>
<td>3</td>
<td>0,01</td>
<td>31/34</td>
<td></td>
</tr>
<tr>
<td>gCD_V1</td>
<td>24</td>
<td>2</td>
<td>37</td>
<td>4</td>
<td>0,01</td>
<td>31/34</td>
<td></td>
</tr>
<tr>
<td>gCD_V2</td>
<td>24</td>
<td>2</td>
<td>37</td>
<td>5</td>
<td>0,01</td>
<td>31/34</td>
<td></td>
</tr>
<tr>
<td>Power_V0</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>0,01</td>
<td>12/14</td>
<td></td>
</tr>
<tr>
<td>Power_V1</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>0,01</td>
<td>12/14</td>
<td></td>
</tr>
<tr>
<td>Power_V2</td>
<td>5</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>0,01</td>
<td>12/14</td>
<td></td>
</tr>
<tr>
<td>Power_V3</td>
<td>5</td>
<td>2</td>
<td>11</td>
<td>2</td>
<td>0,01</td>
<td>21/24</td>
<td></td>
</tr>
<tr>
<td>Power_V4</td>
<td>5</td>
<td>2</td>
<td>11</td>
<td>5</td>
<td>0,01</td>
<td>21/24</td>
<td></td>
</tr>
<tr>
<td>Power_V5</td>
<td>5</td>
<td>2</td>
<td>11</td>
<td>2</td>
<td>0,01</td>
<td>21/24</td>
<td></td>
</tr>
<tr>
<td>sumPower_V0</td>
<td>10</td>
<td>1</td>
<td>13</td>
<td>3</td>
<td>0,01</td>
<td>23/22</td>
<td></td>
</tr>
<tr>
<td>sumPower_V1</td>
<td>10</td>
<td>1</td>
<td>13</td>
<td>3</td>
<td>0,01</td>
<td>23/22</td>
<td></td>
</tr>
<tr>
<td>sumPower_V2</td>
<td>10</td>
<td>1</td>
<td>13</td>
<td>2</td>
<td>0,01</td>
<td>23/22</td>
<td></td>
</tr>
<tr>
<td>sumPower_V3</td>
<td>10</td>
<td>2</td>
<td>21</td>
<td>3</td>
<td>0,01</td>
<td>34/43</td>
<td></td>
</tr>
<tr>
<td>sumPower_V4</td>
<td>10</td>
<td>2</td>
<td>21</td>
<td>5</td>
<td>0,01</td>
<td>34/43</td>
<td></td>
</tr>
<tr>
<td>sumPower_V5</td>
<td>10</td>
<td>2</td>
<td>21</td>
<td>8</td>
<td>0,01</td>
<td>34/43</td>
<td></td>
</tr>
<tr>
<td>Data_V1</td>
<td>21</td>
<td>2</td>
<td>34</td>
<td>7</td>
<td>0,02</td>
<td>59/47</td>
<td></td>
</tr>
<tr>
<td>Data_V2</td>
<td>21</td>
<td>2</td>
<td>34</td>
<td>4</td>
<td>0,06</td>
<td>55/45</td>
<td></td>
</tr>
<tr>
<td>Data_V3</td>
<td>21</td>
<td>2</td>
<td>34</td>
<td>5</td>
<td>0,01</td>
<td>61/49</td>
<td></td>
</tr>
<tr>
<td>Data_V4</td>
<td>21</td>
<td>2</td>
<td>34</td>
<td>5</td>
<td>0,01</td>
<td>61/49</td>
<td></td>
</tr>
<tr>
<td>Data_V5</td>
<td>21</td>
<td>2</td>
<td>34</td>
<td>2</td>
<td>0,01</td>
<td>58/47</td>
<td></td>
</tr>
<tr>
<td>Data_V6</td>
<td>21</td>
<td>2</td>
<td>34</td>
<td>5</td>
<td>0,01</td>
<td>59/47</td>
<td></td>
</tr>
<tr>
<td>Data_V7</td>
<td>21</td>
<td>2</td>
<td>34</td>
<td>4</td>
<td>0,01</td>
<td>59/47</td>
<td></td>
</tr>
</tbody>
</table>
Remarks

- Debugging using constraints is feasible for smaller programs (e.g., at the method level)
- Pre and post conditions can be easily integrated as well as loop invariants
- The quality of the results (e.g., number of statements) depend on the underlying model
Remarks (cont.)

• In order to distinguish diagnosis candidates new knowledge is necessary:
  – Knowledge about intermediate values
  – Specification knowledge
  – New test cases!
DISTINGUISHING TEST CASES
Motivation

• In Vidroha Debroy and W. Eric Wong. Using mutation to automatically suggest fixes for faulty programs, ICST 2010, the authors introduce the notation of possible fixes.
• There might be many of them!
• How to minimize the number of possible fixes?
Motivation (cont.)

1. begin
2. \( i = 2 \times x; \)
3. \( j = 2 \times y; \)
4. \( o1 = i + j; \)
5. \( o2 = i \times i; \)
6. end;

\( x = 1, y = 2, o1 = 8, o2 = 4 \)

Debugger

Diagnosis candidates: 3. \( j = 2 \times y \) and 4. \( o1 = i + j \)

How to distinguish the diagnosis candidates?
Distinguishing test cases

• Use new (distinguishing) test cases for removing diagnosis candidates!

• Note:
  – A diagnosis candidate can be eliminated if the new test case is in contradiction with its behavior.
  – Hence, we compute distinguishing test cases for each pair of candidates and ask the user (or another oracle) for the expected output values.
  – The problem of distinguishing diagnosis candidates is reduced to the problem of computing distinguishing test cases!
Some definitions

\[\Pi\]... Program written in any programming language

**Variable environment** is a set of tuples \((x, v)\) where \(x\) is a variable and \(v\) is its value

\([\Pi](I)\) ... Execution of \(\Pi\) on input environment \(I\)

\([\Pi](I) \supseteq O \iff \Pi\) passes test case\((I, O)\)

\(\neg(\Pi\) passes test case\((I, O)) \iff \Pi\) fails test case\((I, O)\)
Def. distinguishing test case

Given programs $\Pi$ and $\Pi'$. A test case $(I, \emptyset)$ is a distinguishing test case if and only if there is at least one output variable where the value computed when executing $\Pi$ is different from the value computed when executing $\Pi'$ on the same input $I$.

$$(I, \emptyset) \text{ is distinguishing } \Pi \text{ from } \Pi' \iff \exists x : (x, v) \in [\Pi](I) \land (x, v') \in [\Pi'](I) \land v \neq v'$$
Example (cont.)

1. begin
2. \( i = 2 \times x; \)
3. \( j = 3 \times y; \)
4. \( o1 = i + j; \)
5. \( o2 = i \times i; \)
6. end;

Original test case:

Distinguishing test case:

\[ o1 = 5, \quad o2 = 4 \]

\[ x = 1, \quad y = 1 \]

\[ x = 1, \quad y = 2, \quad o1 = 7, \quad o2 = 4 \]

\[ o1 = 6, \quad o2 = 4 \]
Computing distinguishing test cases

• Given two programs.
  1. Convert programs into their constraint representation
  2. Add constraints stating that the inputs have to be equivalent
  3. Add constraints stating that at least one output has to be different
  4. Use the constraint solver to compute the distinguishing test case
Inputs: Two programs $\Pi_1$ and $\Pi_2$ having the same input variables ($IN$) and output variables ($OUT$), and a maximum number of iterations $#I_t$.

Outputs: A distinguishing test case.

1. Call $\text{convert}(\Pi_1, #I_t)$ and store the result in $M_1$.
2. Call $\text{convert}(\Pi_2, #I_t)$ and store the result in $M_2$.
3. Rename all variables $x$ used in constraints $M_1$ to $x_{.P1}$.
4. Rename all variables $x$ used in constraints $M_2$ to $x_{.P2}$.
5. Let $M$ be $M_1 \cup M_2$.
6. For all input variables $x \in IN$ do:
   1. Add the constraint $x_{.P1} = x_{.P2}$ to $M$.
7. For all output variables $x \in OUT$ do:
   1. Add the constraint $x_{.P1} \neq x_{.P2}$ to $M$.
8. Return the values of the input variables obtained when calling a constraint solver on $M$ as result.
## Experimental results

<table>
<thead>
<tr>
<th>Name</th>
<th>It</th>
<th>Var$_{\Pi}$</th>
<th>LOC$_{\Pi}$</th>
<th>Inputs</th>
<th>Outputs</th>
<th>LOC$_{SSA}$</th>
<th>CO</th>
<th>Var$_{CO}$</th>
<th>Diag</th>
<th>Diag$_{nek}$</th>
<th>#UI</th>
<th>Diag$_{TC}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>DivATC_V1</td>
<td>2</td>
<td>5</td>
<td>21</td>
<td>2</td>
<td>1</td>
<td>32</td>
<td>33</td>
<td>29</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DivATC_V2</td>
<td>2</td>
<td>5</td>
<td>21</td>
<td>2</td>
<td>1</td>
<td>32</td>
<td>33</td>
<td>29</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DivATC_V3</td>
<td>2</td>
<td>5</td>
<td>21</td>
<td>2</td>
<td>1</td>
<td>32</td>
<td>33</td>
<td>29</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DivATC_V4</td>
<td>2</td>
<td>5</td>
<td>21</td>
<td>2</td>
<td>1</td>
<td>32</td>
<td>33</td>
<td>29</td>
<td>4</td>
<td>4</td>
<td>1/2</td>
<td>3(1)/1</td>
</tr>
<tr>
<td>GodATC_V1</td>
<td>2</td>
<td>6</td>
<td>35</td>
<td>2</td>
<td>1</td>
<td>49</td>
<td>61</td>
<td>46</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>GodATC_V2</td>
<td>2</td>
<td>6</td>
<td>35</td>
<td>2</td>
<td>1</td>
<td>49</td>
<td>61</td>
<td>46</td>
<td>10</td>
<td>3</td>
<td>1/2/3/4/5</td>
<td>3/3/2/2/1</td>
</tr>
<tr>
<td>GodATC_V3</td>
<td>2</td>
<td>6</td>
<td>35</td>
<td>2</td>
<td>1</td>
<td>49</td>
<td>61</td>
<td>46</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MultiATC_V1</td>
<td>2</td>
<td>5</td>
<td>16</td>
<td>2</td>
<td>1</td>
<td>26</td>
<td>24</td>
<td>19</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MultiATC_V2</td>
<td>2</td>
<td>5</td>
<td>16</td>
<td>2</td>
<td>1</td>
<td>26</td>
<td>24</td>
<td>19</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MultiATC_V3</td>
<td>2</td>
<td>5</td>
<td>16</td>
<td>2</td>
<td>1</td>
<td>26</td>
<td>24</td>
<td>19</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MultiATC_V4</td>
<td>2</td>
<td>5</td>
<td>16</td>
<td>2</td>
<td>1</td>
<td>26</td>
<td>24</td>
<td>19</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MultiV2ATC_V1</td>
<td>2</td>
<td>6</td>
<td>20</td>
<td>2</td>
<td>1</td>
<td>49</td>
<td>67</td>
<td>46</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MultiV2ATC_V2</td>
<td>2</td>
<td>6</td>
<td>20</td>
<td>2</td>
<td>1</td>
<td>49</td>
<td>67</td>
<td>46</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>MultiV2ATC_V3</td>
<td>2</td>
<td>6</td>
<td>20</td>
<td>2</td>
<td>1</td>
<td>49</td>
<td>67</td>
<td>46</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SumATC_V1</td>
<td>2</td>
<td>5</td>
<td>18</td>
<td>2</td>
<td>1</td>
<td>27</td>
<td>24</td>
<td>20</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SumATC_V2</td>
<td>2</td>
<td>5</td>
<td>18</td>
<td>2</td>
<td>1</td>
<td>27</td>
<td>24</td>
<td>20</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SumATC_V3</td>
<td>2</td>
<td>5</td>
<td>18</td>
<td>2</td>
<td>1</td>
<td>27</td>
<td>24</td>
<td>20</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SumPowers_V1</td>
<td>2</td>
<td>11</td>
<td>36</td>
<td>3</td>
<td>1</td>
<td>72</td>
<td>87</td>
<td>70</td>
<td>16</td>
<td>6</td>
<td>1/2/3/4</td>
<td>4/4/2/2</td>
</tr>
<tr>
<td>SumPowers_V2</td>
<td>2</td>
<td>11</td>
<td>36</td>
<td>3</td>
<td>1</td>
<td>72</td>
<td>87</td>
<td>70</td>
<td>11</td>
<td>6</td>
<td>1/2</td>
<td>2/1</td>
</tr>
<tr>
<td>SumPowers_V3</td>
<td>2</td>
<td>11</td>
<td>36</td>
<td>3</td>
<td>1</td>
<td>72</td>
<td>87</td>
<td>70</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Binomial</td>
<td>2</td>
<td>37</td>
<td>189</td>
<td>6</td>
<td>3</td>
<td>481</td>
<td>1329</td>
<td>763</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>BinSearch</td>
<td>2</td>
<td>8</td>
<td>37</td>
<td>4</td>
<td>1</td>
<td>66</td>
<td>102</td>
<td>83</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Hamming</td>
<td>2</td>
<td>15</td>
<td>77</td>
<td>2</td>
<td>1</td>
<td>176</td>
<td>266</td>
<td>185</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Data</td>
<td>2</td>
<td>5</td>
<td>40</td>
<td>1</td>
<td>1</td>
<td>45</td>
<td>57</td>
<td>50</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>while Test</td>
<td>2</td>
<td>16</td>
<td>94</td>
<td>4</td>
<td>1</td>
<td>110</td>
<td>147</td>
<td>103</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
### Experimental results

<table>
<thead>
<tr>
<th>Progr.</th>
<th>D</th>
<th>R&lt;sub&gt;orig&lt;/sub&gt;</th>
<th>D&lt;sub&gt;fit&lt;/sub&gt;</th>
<th>R&lt;sub&gt;fit&lt;/sub&gt;</th>
<th>#UI</th>
<th>D&lt;sub&gt;TC&lt;/sub&gt;</th>
<th>R&lt;sub&gt;TC&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcas01</td>
<td>28</td>
<td>77.7</td>
<td>15</td>
<td>88.0</td>
<td>1</td>
<td>14</td>
<td>88.8</td>
</tr>
<tr>
<td>tcas02</td>
<td>26</td>
<td>79.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tcas03</td>
<td>27</td>
<td>78.4</td>
<td>13</td>
<td>89.6</td>
<td>4</td>
<td>9</td>
<td>92.8</td>
</tr>
<tr>
<td>tcas04</td>
<td>25</td>
<td>80.0</td>
<td>14</td>
<td>88.8</td>
<td>1</td>
<td>14</td>
<td>88.8</td>
</tr>
<tr>
<td>tcas05</td>
<td>25</td>
<td>80.0</td>
<td>14</td>
<td>88.8</td>
<td>1</td>
<td>13</td>
<td>89.6</td>
</tr>
<tr>
<td>tcas06</td>
<td>25</td>
<td>80.0</td>
<td>12</td>
<td>90.4</td>
<td>4</td>
<td>11</td>
<td>91.2</td>
</tr>
<tr>
<td>tcas07</td>
<td>9</td>
<td>92.8</td>
<td>5</td>
<td>96.0</td>
<td>1</td>
<td>5</td>
<td>96.0</td>
</tr>
<tr>
<td>tcas08</td>
<td>27</td>
<td>78.4</td>
<td>13</td>
<td>89.6</td>
<td>4</td>
<td>10</td>
<td>92.0</td>
</tr>
<tr>
<td>tcas09</td>
<td>11</td>
<td>91.2</td>
<td>6</td>
<td>95.2</td>
<td>4</td>
<td>4</td>
<td>96.8</td>
</tr>
<tr>
<td>tcas10</td>
<td>29</td>
<td>76.8</td>
<td>18</td>
<td>85.6</td>
<td>1</td>
<td>17</td>
<td>86.4</td>
</tr>
<tr>
<td>tcas11</td>
<td>23</td>
<td>81.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tcas12</td>
<td>23</td>
<td>81.6</td>
<td>13</td>
<td>89.6</td>
<td>1</td>
<td>13</td>
<td>89.6</td>
</tr>
<tr>
<td>tcas13</td>
<td>27</td>
<td>78.4</td>
<td>5</td>
<td>96.0</td>
<td>0</td>
<td>5</td>
<td>96.0</td>
</tr>
<tr>
<td>tcas14</td>
<td>6</td>
<td>95.2</td>
<td>2</td>
<td>98.3</td>
<td>0</td>
<td>2</td>
<td>98.3</td>
</tr>
<tr>
<td>tcas15</td>
<td>24</td>
<td>80.8</td>
<td>12</td>
<td>90.4</td>
<td>1</td>
<td>11</td>
<td>91.2</td>
</tr>
<tr>
<td>tcas16</td>
<td>26</td>
<td>79.2</td>
<td>6</td>
<td>95.2</td>
<td>0</td>
<td>6</td>
<td>95.2</td>
</tr>
<tr>
<td>tcas17</td>
<td>9</td>
<td>92.8</td>
<td>2</td>
<td>98.3</td>
<td>0</td>
<td>2</td>
<td>98.3</td>
</tr>
<tr>
<td>tcas18</td>
<td>9</td>
<td>92.8</td>
<td>2</td>
<td>98.3</td>
<td>0</td>
<td>2</td>
<td>98.3</td>
</tr>
<tr>
<td>tcas19</td>
<td>9</td>
<td>92.8</td>
<td>2</td>
<td>98.3</td>
<td>0</td>
<td>2</td>
<td>98.3</td>
</tr>
<tr>
<td>tcas20</td>
<td>27</td>
<td>78.4</td>
<td>13</td>
<td>89.6</td>
<td>4</td>
<td>11</td>
<td>91.2</td>
</tr>
<tr>
<td>tcas21</td>
<td>27</td>
<td>78.4</td>
<td>16</td>
<td>87.2</td>
<td>4</td>
<td>15</td>
<td>88.0</td>
</tr>
<tr>
<td>tcas22</td>
<td>8</td>
<td>93.6</td>
<td>4</td>
<td>96.8</td>
<td>1</td>
<td>4</td>
<td>96.8</td>
</tr>
<tr>
<td>tcas23</td>
<td>9</td>
<td>92.8</td>
<td>5</td>
<td>96.0</td>
<td>1</td>
<td>5</td>
<td>96.0</td>
</tr>
<tr>
<td>tcas24</td>
<td>24</td>
<td>80.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tcas25</td>
<td>9</td>
<td>92.8</td>
<td>5</td>
<td>96.0</td>
<td>1</td>
<td>4</td>
<td>96.8</td>
</tr>
<tr>
<td>tcas26</td>
<td>25</td>
<td>80.0</td>
<td>14</td>
<td>88.8</td>
<td>1</td>
<td>14</td>
<td>88.8</td>
</tr>
<tr>
<td>tcas27</td>
<td>25</td>
<td>80.0</td>
<td>12</td>
<td>90.4</td>
<td>1</td>
<td>12</td>
<td>90.4</td>
</tr>
<tr>
<td>tcas28</td>
<td>14</td>
<td>88.8</td>
<td>7</td>
<td>94.4</td>
<td>1</td>
<td>7</td>
<td>94.4</td>
</tr>
<tr>
<td>tcas29</td>
<td>10</td>
<td>92.0</td>
<td>5</td>
<td>96.0</td>
<td>1</td>
<td>5</td>
<td>96.0</td>
</tr>
<tr>
<td>tcas30</td>
<td>13</td>
<td>92.0</td>
<td>9</td>
<td>92.8</td>
<td>1</td>
<td>9</td>
<td>92.8</td>
</tr>
<tr>
<td>tcas31</td>
<td>24</td>
<td>80.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tcas32</td>
<td>23</td>
<td>81.6</td>
<td>13</td>
<td>89.6</td>
<td>1</td>
<td>13</td>
<td>89.6</td>
</tr>
<tr>
<td>tcas33</td>
<td>9</td>
<td>92.8</td>
<td>3</td>
<td>97.0</td>
<td>0</td>
<td>3</td>
<td>97.5</td>
</tr>
<tr>
<td>tcas34</td>
<td>22</td>
<td>82.4</td>
<td>12</td>
<td>90.4</td>
<td>1</td>
<td>12</td>
<td>90.4</td>
</tr>
<tr>
<td>tcas35</td>
<td>14</td>
<td>88.8</td>
<td>8</td>
<td>93.6</td>
<td>1</td>
<td>8</td>
<td>93.6</td>
</tr>
<tr>
<td>tcas36</td>
<td>2</td>
<td>98.4</td>
<td>2</td>
<td>98.4</td>
<td>1</td>
<td>2</td>
<td>98.4</td>
</tr>
<tr>
<td>tcas37</td>
<td>9</td>
<td>92.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tcas38</td>
<td>1</td>
<td>99.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tcas39</td>
<td>9</td>
<td>92.8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tcas40</td>
<td>8</td>
<td>93.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tcas41</td>
<td>27</td>
<td>78.4</td>
<td>17</td>
<td>86.4</td>
<td>1</td>
<td>17</td>
<td>86.4</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td></td>
<td><strong>85.8</strong></td>
<td></td>
<td><strong>92.7</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Remarks

• Computing distinguishing test cases from constraints is possible
• Impact for debugging
• Allows extending test suites

• **But:** Require mutants for each fault candidate computed using model-based debugging
Specification knowledge

• Pre and post conditions
• Invariants
  – Loop invariants
  – Class invariants

• Can be used for improving debugging of loops, recursive functions, and function calls
Loop invariants & more

• Given the following program:

```c
int power(int a, int exp)
PRE: { a ≥ 0, exp ≥ 0 }
1. int e = exp;
2. int res = 1;
3. while (e > 0) {
    INV: { res = a^{exp-e} }
    4. res = res * a;
    5. e = e - 1;
}
6. return res;
POST: { res = a^{exp} }
```
Loop invariants & more (cont)

int power_SSA(int a, int exp) {
    a >= 0 && exp >= 0;
    1. int e_0 = exp;
    2. int res_0 = 1;
    3. bool cond_0 = (e_0 > 0);
    4. int res_1 = res_0 * a;
    5. int e_1 = e_0 - 1;
    res_1 == a^(exp-e_1);
    6. bool cond_1 = cond_0 && (e_1 > 0);
    7. int res_2 = res_1 * a;
    8. int e_2 = e_1 - 1;
    res_2 == a^(exp-e_2);
    ...
    11. int res_4 = \Phi(res_3, res_0, cond_0);
    ...
    res_4 == a^exp;
Intermediate observation

- Pre and post conditions as well as invariants can be easily integrated in the SSA representation (and therefore also the constraint representation).
Handling large programs

• But how to debug larger programs using constraints?

  – Summarizing all constraints -> large constraint representation to be solved!

  – Use pre and post conditions instead of the constraints of methods -> modularization possible!
Modularized debugging

• **Idee**: replace every function call where the pre and post conditions are available with pre && post

```c
foo () {
    int a = 2;
    int exp = 4;
    int result = power(a,exp);
}
```

```c
foo () {
    int a = 2;
    int exp = 4;
    a >= 0 && exp >= 0 && result = a^exp;
}
```
Another observation

• When considering pre and post conditions the problem of debugging even for larger programs is feasible!
Summary specification knowledge

• Specification knowledge is important for debugging
  – Reduce the model size used for debugging
  – Gain information that helps to remove fault candidates

• Integration of specification knowledge into the constraint representation is straightforward
Summary

• Constraints for testing and debugging
• Able to remove up to 93% of the source code for imperative programs on average using filtering and distinguishing test cases.
• Able to remove 99% of statements for combinatorial circuits/programs and 97% for sequential circuits/programs
• Better results than other approaches but computationally more demanding!
Conclusions

• Model-based debugging ensures optimal results
• For small programs (methods,..)
• Allows combining testing and fault localization under one general framework
• There is no silver bullet!
Open challenges

• Combining different debugging approaches
  – Spectrum-based
  – Mutation-based
  – Dependency-based
  – Model-based
  – ...

• Improving performance

• Handling OO constructs still open research question
Some papers...


...and some more...

END OF PART 2

QUESTIONS?