Automated Error Localization and Correction for Imperative Programs

Robert Könighofer and Roderick Bloem
IAIK – Graz University of Technology

robert.koenighofer@iaik.tugraz.at
www.iaik.tugraz.at

This work was supported in part by the European Commission through project DIAMOND (FP7-2009-IST-4-248613).
Outline

- Addressed problem
- Debugging approach
  - Program analysis
  - Error localization
  - Error correction
- Experimental results
- Summary and conclusions
Addressed Problem

- Automate debugging of simple software

Input:
- Incorrect program
- Specification:
  - Assertions
  - Reference Implementation

Output:
- Potential error locations
- Potential repairs

```c
int max(int x, int y) {
    int res = x;
    if(y > x)
        res = x;
    assert(res>=x && res>=y);
    return res;
}
```
Debugging Approach

- Incorrect Program
- Pre-Processing
- Program Analysis
- Diagnostic Data
- Error Localization
- Error Correction
- Diagnoses
- Repairs

Symbolic Execution
SMT Solving
SMT Solving
Debugging Approach

Incorrect Program -> Pre-Processing -> Program Analysis

Diagnostic Data

Error Localization -> Diagnoses

Error Correction -> Repairs

Symbolic Execution

SMT Solving

SMT Solving
Debugging Approach: Pre-Processing

- Identification of Components
  - Right-hand side (RHS) of assignments
  - Efficient program analysis

```c
int max(int x, int y) {
    int res = x;
    if(y > x)
        res = x;
    assert(res>=x && res>=y);
    return res;
}
```
Debugging Approach: Pre-Processing

- Identification of Components
  - Right-hand side (RHS) of assignments
  - Efficient program analysis

```c
int max(int x, int y) {
    int res = x; \[c_0\]
    if(y > x) {
        res = x; \[c_1\]
        assert(res>=x && res>=y);
    }
    return res;
}
```
Debugging Approach:
Pre-Processing

- Identification of Components
  - Right-hand side (RHS) of assignments
  - Efficient program analysis
  - Can handle all incorrect expressions

```c
int max(int x, int y) {
    int res = x; \[c_0\]
    if(y > x)
        res = x; \[c_1\]
    assert(res>=x && res>=y);
    return res;
}
```
Debugging Approach:

Pre-Processing

- Components may be faulty
- Transformation:

```c
// AC = ‘Assume Correct’
bool AC_c0;
bool AC_c1;

int max(int x, int y) {
    int res = AC_c0 ? x : repair_c0(x,y);
    if(y > x)
        res = AC_c1 ? x : repair_c1(x,y,res);
    assert(res>=x && res>=y);
    return res;
}
```

```c
int max(int x, int y) {
    int res = x;
    if(y > x)
        res = x;
    assert(res>=x && res>=y);
    return res;
}
```
When is my buggy program correct?
We use Symbolic Execution
- Allows incomplete analysis

```c
int max(int x, int y) {
    int res = x;
    if(y > x)
        res = x;
    assert(res>=x && res>=y);
    return res;
}
```

Correct \( X, Y \) = \( Y \leq X \)
Debugging Approach:

Program Analysis

- When is my program correct?

```c
bool AC_c0; // AC = ‘Assume Correct’
bool AC_c1;
int max(int x, int y) {
    int res = AC_c0 ? x : repair_c0(x,y);
    if(y > x)
        res = AC_c1 ? x : repair_c1(x,y, res);
    assert(res>=x && res>=y);
    return res;
}
```

\[
Correct(X, Y, R_0, R_1, AC_c0, AC_c1) = \\
\exists A_0, A_1: \\
[Y > X \land A_1 \geq X \land A_1 \geq Y \lor \\
Y \leq X \land A_0 \geq X \land A_0 \geq Y] \land \\
A_0 = AC_c0 ? X : R_0 \land \\
A_1 = AC_c1 ? X : R_1
\]

\[
R_0 = repair_c0(X, Y) \\
R_1 = repair_c1(X, Y, X)
\]
Debugging Approach: Program Analysis

- When is my buggy program correct?

```c
bool AC_c0; // AC = 'Assume Correct'
bool AC_c1;

int max(int x, int y) {
    int res = AC_c0 ? x : repair_c0(x,y);
    if(y > x)
        res = AC_c1 ? x : repair_c1(x,y,res);
    assert(res>=x && res>=y);
    return res;
}
```

Correct \( (X,Y,R_0,R_1,AC_c_0,AC_c_1) = \exists A_0,A_1:\)
\[
\begin{align*}
Y &> X \land A_1 \geq X \land A_1 \geq Y \lor \\
Y &\leq X \land A_0 \geq X \land A_0 \geq Y \lor \\
A_0 &= AC_c_0 ? X : R_0 \land \\
A_1 &= AC_c_1 ? X : R_1
\end{align*}
\]

\[
R_0 = repair_c_0(X,Y) \\
R_1 = repair_c_1(X,Y,X)
\]

Diagnostic Data
Debugging Approach

Incorrect Program → Pre-Processing → Program Analysis

Symbolic Execution

Diagnostic Data

Error Localization → Diagnoses
Error Correction → Repairs

SMT Solving

SMT Solving
Debugging Approach

- Incorrect Program
- Pre-Processing
- Program Analysis
- Diagnostic Data
- Error Localization
- Error Correction
- Diagnoses
- Repairs

Symbolic Execution
SMT Solving
SMT Solving
Debugging Approach:

Error Localization

- **Goal:** compute diagnoses
  - Sets of components that
    - … if assumed to be incorrect, make the program repairable
    - … can be modified to make the program correct
    - … may be responsible for the incorrectness
Debugging Approach:

Error Localization

**Goal:** compute diagnoses

Sets of components that
- … if assumed to be incorrect, make the program repairable
- … can be modified to make the program correct
- … may be responsible for the incorrectness

```c
int max(int x, int y) {
    int res = x;
    if(y > x)
        res = y;
    assert(res>=x && res>=y);
    return res;
}
```

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Repairable</th>
<th>Diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>correct</td>
<td>correct</td>
<td>no</td>
</tr>
<tr>
<td>correct</td>
<td>incorrect</td>
<td>yes</td>
</tr>
<tr>
<td>incorrect</td>
<td>correct</td>
<td>no</td>
</tr>
<tr>
<td>incorrect</td>
<td>incorrect</td>
<td>yes</td>
</tr>
</tbody>
</table>
Debugging Approach: Error Localization

- **Goal**: compute diagnoses

Sets of components that
- ... if assumed to be incorrect, make the program repairable
- ... can be modified to make the program correct
- ... may be responsible for the incorrectness

```c
int max(int x, int y) {
    int res = x;
    if(y > x)
        res = x;
    assert(res>=x && res>=y);
    return res;
}
```

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Repairable</th>
<th>Diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>c₀</td>
<td>correct</td>
<td>no</td>
</tr>
<tr>
<td>c₁</td>
<td>correct</td>
<td>yes</td>
</tr>
<tr>
<td>incorrect</td>
<td>correct</td>
<td>no</td>
</tr>
<tr>
<td>incorrect</td>
<td>incorrect</td>
<td>yes</td>
</tr>
</tbody>
</table>
Debugging Approach: Error Localization

- **Goal**: compute diagnoses
  - Sets of components that
    - … if assumed to be incorrect, make the program repairable
    - … can be modified to make the program correct
    - … may be responsible for the incorrectness

```c
int max(int x, int y) {
    int res = x;  // c_0
    if(y > x)
        res = repair(...); // c_1
    assert(res>=x && res>=y);
    return res;
}
```

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Repairable</th>
<th>Diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>correct</td>
<td>correct</td>
<td>no</td>
</tr>
<tr>
<td>correct</td>
<td>incorrect</td>
<td>yes</td>
</tr>
<tr>
<td>incorrect</td>
<td>correct</td>
<td>no</td>
</tr>
<tr>
<td>incorrect</td>
<td>incorrect</td>
<td>yes</td>
</tr>
</tbody>
</table>
Debugging Approach:
Error Localization

- **Goal**: compute diagnoses

Sets of components that
- … if assumed to be incorrect, make the program repairable
- … can be modified to make the program correct
- … may be responsible for the incorrectness

```c
int max(int x, int y) {
    int res = repair(...);
    if(y > x)
        res = x;
    assert(res>=x && res>=y);
    return res;
}
```

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Repairable</th>
<th>Diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>correct</td>
<td>correct</td>
<td>no</td>
</tr>
<tr>
<td>correct</td>
<td>incorrect</td>
<td>yes</td>
</tr>
<tr>
<td>incorrect</td>
<td>correct</td>
<td>no</td>
</tr>
<tr>
<td>incorrect</td>
<td>incorrect</td>
<td>yes</td>
</tr>
</tbody>
</table>

Robert Könighofer  
Austin, 2011-10-31  
FMCAD 2011
Debugging Approach: Error Localization

- **Goal**: compute diagnoses
  Sets of components that
  - … if assumed to be incorrect, make the program repairable
  - … can be modified to make the program correct
  - … may be responsible for the incorrectness

```c
int max(int x, int y) {
    int res = repair(...);
    if(y > x)
        res = repair(...);
    assert(res>=x && res>=y);
    return res;
}
```

<table>
<thead>
<tr>
<th>Assumption</th>
<th>Repairable</th>
<th>Diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>correct</td>
<td>correct</td>
<td>no</td>
</tr>
<tr>
<td>correct</td>
<td>incorrect</td>
<td>yes</td>
</tr>
<tr>
<td>incorrect</td>
<td>correct</td>
<td>no</td>
</tr>
<tr>
<td>incorrect</td>
<td>incorrect</td>
<td>yes</td>
</tr>
</tbody>
</table>
Debugging Approach:

Error Localization

- Checking for Repairability:

```c
bool AC_c0; // AC = ‘Assume Correct’
bool AC_c1;

int max(int x, int y) {
    int res = AC_c0 ? x : repair_c0(x,y);
    if(y > x)
        res = AC_c1 ? x : repair_c1(x,y,res);
    assert(res>=x && res>=y);
    return res;
}
```

**Diagnostic Data:**

\[
Correct(X, Y, R_0, R_1, AC_c0, AC_c1) =
\exists A_0, A_1:
[Y > X \land A_1 \geq X \land A_1 \geq Y \lor
Y \leq X \land A_0 \geq X \land A_0 \geq Y] \land
A_0 = AC_c0 \ ? \ X : \ R_0 \land
A_1 = AC_c1 \ ? \ X : \ R_1
\]
Debugging Approach:

Error Localization

- Checking for Repairability:

```cpp
bool AC_c0; // AC = ‘Assume Correct’
bool AC_c1;
int max(int x, int y) {
    int res = AC_c0 ? x : repair_c0(x,y);
    if(y > x)
        res = AC_c1 ? x : repair_c1(x,y,res);
    assert(res>=x && res>=y);
    return res;
}
```

Diagnostic Data:

\[ \text{Correct}(X, Y, R_0, R_1, AC_c_0, AC_c_1) = \exists A_0, A_1: \]
\[ [Y > X \land A_1 \geq X \land A_1 \geq Y \lor \]
\[ Y \leq X \land A_0 \geq X \land A_0 \geq Y] \land \]
\[ A_0 = AC_c_0 \land X : R_0 \land \]
\[ A_1 = AC_c_1 \land X : R_1 \]

\[ \text{Repairable}(AC_c_0, AC_c_1) = \forall X, Y: \exists R_0, R_1: \]
\[ \text{Correct}(X, Y, R_0, R_1, AC_c_0, AC_c_1) \]
Debugging Approach:

**Error Localization**

\[
\text{Repairable}(AC_c_0, AC_c_1) = \forall X, Y: \exists R_0, R_1: \text{Correct}(X, Y, R_0, R_1, AC_c_0, AC_c_1)
\]

- Use SMT-Solver to check repairability
  - Problem: Quantifier alternation
  - Check for a finite set of input values only
  - False positives
Debugging Approach:

Error Localization

- Use SMT-Solver to check repairability
  - Problem: Quantifier alternation
  - Check for a finite set of input values only
  - False positives
- A set D is a diagnosis iff
  \[ \text{Repairable}(AC_{c_0}, AC_{c_1}, \ldots, AC_{c_n}) = \text{true} \]
  with \( AC_{c_k} = \text{false} \) iff \( c_k \in D \)
- Computing diagnoses naively: check every set
- Our algorithm: Unsatisfiable cores + hitting sets
Debugging Approach

- Incorrect Program
- Pre-Processing
- Program Analysis
- Diagnostic Data
- Error Localization
- Diagnoses
- Error Correction
- Repairs

Symbolic Execution
SMT Solving
SMT Solving
Debugging Approach:

Error Correction

- We know which components are faulty
  - E.g.: $D_1 = \{c_8\}$, $D_2 = \{c_2, c_4\}$

- Error Correction
  - Compute replacements for the faulty components
  - E.g.: $c_8 \rightarrow x+9$
  - $c_2 \rightarrow a \leq b$ and $c_4 \rightarrow 1$
Debugging Approach: Error Correction

- Use information about faulty components:

\[
\text{Correct}(X, Y, R_0, R_1, AC_{c_0}, AC_{c_1}) = \\
\exists A_0, A_1: \\
[Y > X \land A_1 \geq X \land A_1 \geq Y \lor \\
Y \leq X \land A_0 \geq X \land A_0 \geq Y] \land \\
A_0 = AC_{c_0} ? X : R_0 \land \\
A_1 = AC_{c_1} ? X : R_1
\]

\[
R_0 = \text{repair}_{c_0}(X, Y) \\
R_1 = \text{repair}_{c_1}(X, Y, X)
\]

Diagnostic Data

c_{1}=\text{incorrect}

\[
\text{Correct}'(X, Y) = \\
\exists A_0, A_1: \\
[Y > X \land A_1 \geq X \land A_1 \geq Y \lor \\
Y \leq X \land A_0 \geq X \land A_0 \geq Y] \land \\
A_0 = X \\
A_1 = \text{repair}_{c_1}(X, Y, X)
\]

Simplified

\[
\text{Correct}'(X, Y) = Y \leq X \lor \\
\text{repair}_{c_1}(X, Y, X) \geq X \land \\
\text{repair}_{c_1}(X, Y, X) \geq Y
\]
Debugging Approach:

**Error Correction**

\[ \text{Correct}'(X, Y) = Y \leq X \lor \text{repair}_c_1(X, Y, X) \geq X \land \text{repair}_c_1(X, Y, X) \geq Y \]

- **Goal**: make \( \text{Correct}'(\ldots) \) equal to true
Debugging Approach:

Error Correction

\[ Correct'(X, Y) = Y \leq X \lor repair_c_1(X, Y, X) \geq X \land repair_c_1(X, Y, X) \geq Y \]

- **Goal**: make \( Correct'(\ldots) \) equal to true
- Simpler problem:
  - Checking a given expression is easy
  - Example: \( repair_c_1(x, y, res) = x + 4 \)

\[ Correct'(X, Y) = Y \leq X \lor repair_c_1(X, Y, X) \geq X \land repair_c_1(X, Y, X) \geq Y \]

\[ Correct''(X, Y) = Y \leq X \lor X + 4 \geq X \land X + 4 \geq Y \]

- \( Correct''(0, 5) = \text{false} \rightarrow 'x + 4' \) is not a valid repair
- Solver can also return a counterexample
Debugging Approach:

**Error Correction**

- Finding a substitution for `repair()` is more difficult
- **Idea**: Replace `repair(…)` with a template for expressions
- E.g.: `repair(x,y,res) = k_0 + k_1*x + k_2*y + k_3*res`
Debugging Approach:

Error Correction

- Finding a substitution for `repair()` is more difficult
- **Idea**: Replace `repair(...)` with a template for expressions
- E.g.: `repair(x,y,res) = k_0 + k_1*x + k_2*y + k_3*res`

\[
Correct'(X,Y) = Y \leq X \lor \text{repair}_c_1(X,Y,X) \geq X \land \text{repair}_c_1(X,Y,X) \geq Y
\]

\[
Correct''(X,Y,k_0,k_1,k_2,k_3) = Y \leq X \lor k_0 + k_1 \cdot X + k_2 \cdot Y + k_3 \cdot X \geq X \land k_0 + k_1 \cdot X + k_2 \cdot Y + k_3 \cdot X \geq Y
\]

- Find \( k_0, k_1, k_2, k_3 \) such that \( Correct''(X, Y, k_0, k_1, k_2, k_3) \) holds for all \( X, Y \)
- Now: find parameter values instead of expressions
Debugging Approach:

Error Correction

- Solvers cannot handle conditions like

\[ \exists k_0, k_1, k_2, k_3 : \forall X, Y: \text{Correct}''(X, Y, k_0, k_1, k_2, k_3) \]

efficiently

- Can find \( k_0, k_1, k_2, k_3 \) such that \( \text{Correct}(X, Y, k_0, k_1, k_2, k_3) \) is true for \textbf{some} inputs \( X, Y \).

- Can compute repairs that are correct for \textbf{some} inputs
Debugging Approach:

Error Correction

- Iterative repair refinement:

  ![Diagram showing the iterative repair refinement process with a database, counterexample, check if correct, and repair steps.]

  **No Repair Exists**
  \[\text{Compute Repair} \rightarrow \text{Check if Correct} \rightarrow \text{Repair} \rightarrow \text{Correct Repair} \rightarrow \text{Done} \rightarrow \text{Fail}\]

  **Input Database**

  See also:
Debugging Approach

Incorrect Program → Pre-Processing → Program Analysis → Diagnostic Data → Error Localization → Diagnoses → Error Correction → Repairs

- Diagnoses
- Repairs
- Error Correction
- Error Localization
- Diagnostic Data
- Program Analysis
- Pre-Processing
- Incorrect Program
Implementation

- For simple C programs
- Tool: FoREnSiC
  - Various debugging methods
- Program analysis:
  - Concolic execution
  - Extension of CREST
- SMT-Solver:
  - Yices and Z3
  - Linear Integer Arithmetic
  - Bitvector Arithmetic under Development
Experimental Results:

Comparison with Sketch [1]

- TCAS example [2] (180 LOC, 12 inputs, 44 components):

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tcas2</td>
<td>2</td>
<td>70</td>
<td>2</td>
<td>271</td>
<td>11</td>
<td></td>
<td>75</td>
<td>29</td>
<td>1</td>
<td>297</td>
</tr>
<tr>
<td>tcas7</td>
<td>2</td>
<td>123</td>
<td>5</td>
<td>42</td>
<td>10</td>
<td></td>
<td>70</td>
<td>20</td>
<td>4</td>
<td>1459</td>
</tr>
<tr>
<td>tcas8</td>
<td>2</td>
<td>122</td>
<td>5</td>
<td>37</td>
<td>10</td>
<td></td>
<td>79</td>
<td>6.3</td>
<td>1</td>
<td>42</td>
</tr>
<tr>
<td>tcas16</td>
<td>2</td>
<td>123</td>
<td>5</td>
<td>43</td>
<td>10</td>
<td></td>
<td>72</td>
<td>22</td>
<td>2</td>
<td>764</td>
</tr>
<tr>
<td>tcas17</td>
<td>2</td>
<td>125</td>
<td>5</td>
<td>41</td>
<td>10</td>
<td></td>
<td>70</td>
<td>11</td>
<td>2</td>
<td>666</td>
</tr>
<tr>
<td>tcas18</td>
<td>2</td>
<td>124</td>
<td>5</td>
<td>38</td>
<td>10</td>
<td></td>
<td>75</td>
<td>9.3</td>
<td>2</td>
<td>744</td>
</tr>
<tr>
<td>tcas19</td>
<td>2</td>
<td>123</td>
<td>5</td>
<td>40</td>
<td>10</td>
<td></td>
<td>70</td>
<td>9.6</td>
<td>1</td>
<td>42</td>
</tr>
<tr>
<td>tcas36</td>
<td>0</td>
<td>3.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td>44</td>
<td>7.0</td>
<td>4</td>
<td>796</td>
</tr>
</tbody>
</table>


Experimental Results:

Comparison with Sketch [1]

- TCAS example [2] (180 LOC, 12 inputs, 44 components):

<table>
<thead>
<tr>
<th>Diagnosis</th>
<th>Repair</th>
<th>Sketch (8 bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Found</td>
<td>Time [sec]</td>
</tr>
<tr>
<td>tcas2</td>
<td>2</td>
<td>70</td>
</tr>
<tr>
<td>tcas7</td>
<td>2</td>
<td>123</td>
</tr>
<tr>
<td>tcas8</td>
<td>2</td>
<td>122</td>
</tr>
<tr>
<td>tcas16</td>
<td>2</td>
<td>123</td>
</tr>
<tr>
<td>tcas17</td>
<td>2</td>
<td>125</td>
</tr>
<tr>
<td>tcas18</td>
<td>2</td>
<td>124</td>
</tr>
<tr>
<td>tcas19</td>
<td>2</td>
<td>123</td>
</tr>
<tr>
<td>tcas36</td>
<td>0</td>
<td>3.0</td>
</tr>
</tbody>
</table>

- Do not take too seriously; different solvers, objectives, …
Experimental Results:

Comparison with Sketch [1]

- TCAS example [2] (180 LOC, 12 inputs, 44 components):

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>tcas2</td>
<td>2</td>
<td>70</td>
<td>2</td>
<td>271</td>
<td>11</td>
<td>75</td>
<td>29</td>
<td>1</td>
<td>297</td>
</tr>
<tr>
<td>tcas7</td>
<td>2</td>
<td>123</td>
<td>5</td>
<td>42</td>
<td>10</td>
<td>70</td>
<td>20</td>
<td>4</td>
<td>1459</td>
</tr>
<tr>
<td>tcas8</td>
<td>2</td>
<td>122</td>
<td>5</td>
<td>37</td>
<td>10</td>
<td>79</td>
<td>6.3</td>
<td>1</td>
<td>42</td>
</tr>
<tr>
<td>tcas16</td>
<td>2</td>
<td>123</td>
<td>5</td>
<td>43</td>
<td>10</td>
<td>72</td>
<td>22</td>
<td>2</td>
<td>764</td>
</tr>
<tr>
<td>tcas17</td>
<td>2</td>
<td>125</td>
<td>5</td>
<td>41</td>
<td>10</td>
<td>70</td>
<td>11</td>
<td>2</td>
<td>666</td>
</tr>
<tr>
<td>tcas18</td>
<td>2</td>
<td>124</td>
<td>5</td>
<td>38</td>
<td>10</td>
<td>75</td>
<td>9.3</td>
<td>2</td>
<td>744</td>
</tr>
<tr>
<td>tcas19</td>
<td>2</td>
<td>123</td>
<td>5</td>
<td>40</td>
<td>10</td>
<td>70</td>
<td>9.6</td>
<td>1</td>
<td>42</td>
</tr>
<tr>
<td>tcas36</td>
<td>0</td>
<td>3.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>44</td>
<td>7.0</td>
<td>4</td>
<td>796</td>
</tr>
</tbody>
</table>

- Accurate diagnoses and correct repairs in reasonable time!
Experimental Results:

Example: tcas35

- **Correct:**
  
  \[
  \text{ClimbIn} \ ? \text{UpSep+100} : \text{UpSep}
  \]

- **Buggy:**

  \[
  \underbrace{\text{ClimbIn} \ ? \text{UpSep}} \ : \underbrace{\text{UpSep+100}}
  \]

- **Diagnosis:** \{c_5,c_6\}
Experimental Results:

**Example: tcas35**

- **Correct:**
  - Diagnosis: \( \{c_5, c_6\} \)
  - Repair:

- **Buggy:**

<table>
<thead>
<tr>
<th>It.</th>
<th>( c_5 )</th>
<th>( c_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>728</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>-ClimbIn+1000</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>-ClimbIn+58</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>UpSep+352</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>UpSep-1000</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>UpSep+100</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>UpSep</td>
<td>-OtherTrAlt+11</td>
</tr>
<tr>
<td>9</td>
<td>UpSep+1</td>
<td>UpSep-1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>It.</th>
<th>( c_5 )</th>
<th>( c_6 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>UpSep+2</td>
<td>CurVerSep-795</td>
</tr>
<tr>
<td>11</td>
<td>UpSep+3</td>
<td>3114</td>
</tr>
<tr>
<td>12</td>
<td>UpSep+100</td>
<td>UpSep-1000</td>
</tr>
<tr>
<td>13</td>
<td>UpSep+100</td>
<td>-OtherTrAlt+11</td>
</tr>
<tr>
<td>14</td>
<td>UpSep+4</td>
<td>CurVerSep</td>
</tr>
<tr>
<td>15</td>
<td>UpSep+100</td>
<td>3114</td>
</tr>
<tr>
<td>16</td>
<td>UpSep+5</td>
<td>UpSep+1000</td>
</tr>
<tr>
<td>17</td>
<td>UpSep+100</td>
<td>OwnTrAlt</td>
</tr>
<tr>
<td>18</td>
<td>UpSep+100</td>
<td>UpSep</td>
</tr>
</tbody>
</table>
Summary and Conclusion

- **Problem**: debug incorrect programs
- **Debugging method**: 

  ![Diagram of the debugging process]

- Works nicely, but a lot of room for improvement!