ON-THE-FLY DYNAMIC DEAD VARIABLE ANALYSIS

Eric Mercer
Joel Self
Software Model Checking Lab
Brigham Young University
SPIN 2007
Berlin, Deutschland
February 25, 1991 a patriot missile failed to intercept an incoming Iraqi scud missile killing 28 solders in a military barrack.
Computer System Glitches

- 1991 Patriot missile clock drift
- 1994 Intel FPDIV
- 1996 Ariane 5 Rocket Explosion
- 2000 I Love You Virus
- 2004 BMW Engine Stall
- 2006 Utah voting machines fail
- 2006 Segway Destabilize
- 2007 Encryption broken
- Blu-Ray HD DVD

Eric Mercer & Joel Self  3/12/09
More Glitches

- 02/2007 F-22 Raptor system crash
- 02/2007 Subliminal slot machine messages
- 02/2007 Dow Jones plunge
What Can Be Done?

- Software errors may be small, but catastrophic
- Traditional testing will miss these small errors
- Model checking can help find subtle errors
- Model checking takes a system and a specification
  - Exhaustively enumerates all behaviors
  - Checks if behaviors meet or violate the specification
- Result is a proof
Example State Space

1: \texttt{f(int a, int b, int c)}
2: \texttt{if(a > 0) then}
3: \hspace{1em} b = 3;
4: \hspace{1em} c = 5;
5: \hspace{1em} \texttt{print a,b,c; assert c != 7}
6: \texttt{end}
Problem With Model Checking

- Size of state space can be prohibitive
- 32 bit integer = $2^{32}$ or around 4 billion values
- Data abstraction can help
- Represent many values with fewer values
- Dead variable analysis
- Precise data abstraction
- Requires no theorem prover
- Builds on known static analysis techniques
Dead Variable Analysis (DVA)

- Label variables live or dead at a location
  - Live = current valuation will be used in some future
  - Dead = current valuation will not be used in any future
- Dead variables do not affect program behavior
- We ignore these valuations
- Dead variable values do not distinguish states
Static Dead Variable Analysis (SDVA)

1: \textbf{f}(\texttt{int} \ a, \ \texttt{int} \ b, \ \texttt{int} \ c)
2: \quad \textbf{if}(a > 0) \ \textbf{then}
3: \quad \quad b = 3;
4: \quad \quad c = 5;
5: \quad \textbf{print} \ a, \ b, \ c;
6: \quad \textbf{end}

Eric Mercer & Joel Self   3/12/09
1: f(int a, int b, int c)
2: if (a > 0) then
3:   b = 3;
4:   c = 5;
5: print a, b, c;
6: end
Static Dead Variable Analysis

1: \texttt{f(int a, int b, int c)}
2: \texttt{if(a > 0) then}
3: \texttt{b = 3;}
4: \texttt{c = 5;}
5: \texttt{print a, b, c;}
6: \texttt{end}
1: \textbf{f}(\textbf{int} \ a, \ \textbf{int} \ b, \ \textbf{int} \ c) \\
2: \quad \textbf{if}(a > 0) \ \textbf{then} \\
3: \quad \quad b = 3; \\
4: \quad \quad c = 5; \\
5: \quad \textbf{print} \ a, \ b, \ c; \\
6: \quad \textbf{end}
Related Work

- M. S. Lewis and M. D. Jones, *A Dead Variable Analysis for Explicit Model Checking*, 2006
Feasible Paths

1: \texttt{f(int a, int b, int c)}
2: \texttt{if(a > 0) then}
3: \texttt{b = 3;}
4: \texttt{c = 5;}
5: \texttt{print a, b, c;}
6: \texttt{end}

a > 0

2
3
4
5

b is dead at location 2

a \leq 0

2
3
4
5

b is live at location 2

b is dead at location 2

Eric Mercer & Joel Self  3/12/09
1: \textbf{f(int a, int b, int c)}
2: \textbf{if}(a > 0) \textbf{then}
3: \hspace{1em}b = 3;
4: \hspace{1em}c = 5;
5: \hspace{1em}\textbf{print} a, b, c;
6: \hspace{1em}\textbf{end}
1: \textbf{f}(\textbf{int} \ a, \textbf{int} \ b, \textbf{int} \ c) \\
2: \quad \textbf{if} (a > 0) \textbf{ then} \\
3: \quad \quad b = 3; \\
4: \quad \quad c = 5; \\
5: \quad \textbf{print} \ a, \ b, \ c; \\
6: \quad \textbf{end}
Previous DDVA Work

- Original DDVA
  - Uses forward analysis
  - Results dependent on depth bound
  - Does not compute maximal reduction
  - Does not handle loops

State Generation Along a Path

Start: depth = 9

10: a is defined

17: a is redefined

18: a is used

Exit: depth = 16
DVA Maximal Reduction

Concrete State Space

Abstract State Space

\[ (\leq_c) \]

Eric Mercer & Joel Self 3/12/09
DVA Maximal Reduction

- Only live if exists concrete trace that requires it to be live

Concrete State Space

\[ \leq_c \]

Abstract State Space

\[ v \text{ is live} \]
Maximal DDVA Implementation

1. Take a full trace through the system
2. Apply \textit{dead} to states in trace to find dead variables
3. Mark dead variables
4. Re-store states in \textit{Visited}
5. Resume model checking
Maximal Dead Variable Analysis

1: \( f(\text{int } a, \text{ int } b, \text{ int } c) \)
2: \( \text{if}(a > 0) \) then
3: \( b = 3; \)
4: \( c = 5; \)
5: \( \text{print } a, b, c; \)
6: \text{end}
Non-Determinism

Location

1: a = get_input();
2: c = get_input();
3: if c > 2 then
4: a = 5;
5: print a, b, c;
## Results Tables

### easy3

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Explore Depth</th>
<th>States Generated</th>
<th>Total Time</th>
<th>User Time</th>
<th>Memory Used (MB)</th>
<th>Abstraction Time</th>
<th>Re-store Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>34640</td>
<td>0m12.764s</td>
<td>0m9.969s</td>
<td>34.50</td>
<td>0.0s</td>
<td>N/A</td>
</tr>
<tr>
<td>Static</td>
<td>N/A</td>
<td>15814</td>
<td>0m06.605s</td>
<td>0m5.336s</td>
<td>33.80</td>
<td>0.001s</td>
<td>N/A</td>
</tr>
<tr>
<td>Original best</td>
<td>2</td>
<td>15814</td>
<td>0m10.765s</td>
<td>0m9.313s</td>
<td>34.46</td>
<td>3.792s</td>
<td>N/A</td>
</tr>
<tr>
<td>Original worst</td>
<td>2</td>
<td>15814</td>
<td>0m10.765s</td>
<td>0m9.313s</td>
<td>34.46</td>
<td>3.792s</td>
<td>N/A</td>
</tr>
<tr>
<td>Maximal</td>
<td>N/A</td>
<td>10330</td>
<td>0m08.105s</td>
<td>0m7.002s</td>
<td>25.5312</td>
<td>2.017s</td>
<td>1.021</td>
</tr>
</tbody>
</table>

### littleBranch

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Explore Depth</th>
<th>States Generated</th>
<th>Total Time</th>
<th>User Time</th>
<th>Memory Used (MB)</th>
<th>Abstraction Time</th>
<th>Re-store Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>864</td>
<td>0m0.442s</td>
<td>0m0.272s</td>
<td>30.90</td>
<td>0.0s</td>
<td>N/A</td>
</tr>
<tr>
<td>Static</td>
<td>N/A</td>
<td>721</td>
<td>0m0.405s</td>
<td>0m0.236s</td>
<td>31.40</td>
<td>0.0010s</td>
<td>N/A</td>
</tr>
<tr>
<td>Original best</td>
<td>6</td>
<td>658</td>
<td>0m0.344s</td>
<td>0m0.280s</td>
<td>31.43</td>
<td>0.0740s</td>
<td>N/A</td>
</tr>
<tr>
<td>Original worst</td>
<td>2</td>
<td>721</td>
<td>0m0.340s</td>
<td>0m0.268s</td>
<td>31.43</td>
<td>0.0492s</td>
<td>N/A</td>
</tr>
<tr>
<td>Maximal</td>
<td>N/A</td>
<td>530</td>
<td>0m0.223s</td>
<td>0m0.176s</td>
<td>23.79</td>
<td>0.0138s</td>
<td>1.391</td>
</tr>
</tbody>
</table>

### multiBranch

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Explore Depth</th>
<th>States Generated</th>
<th>Total Time</th>
<th>User Time</th>
<th>Memory Used (MB)</th>
<th>Abstraction Time</th>
<th>Re-store Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>294515</td>
<td>1m49.170s</td>
<td>1m28.146s</td>
<td>87.10</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Static</td>
<td>N/A</td>
<td>217454</td>
<td>1m21.780s</td>
<td>1m06.084s</td>
<td>74.87</td>
<td>0.002s</td>
<td>N/A</td>
</tr>
<tr>
<td>Original best</td>
<td>16</td>
<td>176651</td>
<td>1m41.458s</td>
<td>1m27.673s</td>
<td>75.79</td>
<td>42.67s</td>
<td>N/A</td>
</tr>
<tr>
<td>Original worst</td>
<td>5</td>
<td>217478</td>
<td>2m10.965s</td>
<td>1m55.179s</td>
<td>83.46</td>
<td>46.35s</td>
<td>N/A</td>
</tr>
<tr>
<td>Maximal</td>
<td>N/A</td>
<td>145440</td>
<td>2m36.640s</td>
<td>2m25.453s</td>
<td>57.99</td>
<td>7.51s</td>
<td>1.06</td>
</tr>
</tbody>
</table>
## Results Tables

### lexer

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Explore Depth</th>
<th>States Generated</th>
<th>Total Time</th>
<th>User Time</th>
<th>Memory Used (MB)</th>
<th>Abstraction Time</th>
<th>Re-store Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>262843</td>
<td>1m28.391s</td>
<td>1m10.220s</td>
<td>66.90</td>
<td>0.0s</td>
<td>N/A</td>
</tr>
<tr>
<td>Static</td>
<td>N/A</td>
<td>226169</td>
<td>1m17.633s</td>
<td>1m01.876s</td>
<td>66.32</td>
<td>0.002s</td>
<td>N/A</td>
</tr>
<tr>
<td>Original best</td>
<td>2</td>
<td>225370</td>
<td>1m51.479s</td>
<td>1m34.442s</td>
<td>71.30</td>
<td>31.66s</td>
<td>N/A</td>
</tr>
<tr>
<td>Original worst</td>
<td>3</td>
<td>226172</td>
<td>1m53.866s</td>
<td>1m36.554s</td>
<td>71.13</td>
<td>33.46s</td>
<td>N/A</td>
</tr>
<tr>
<td>Maximal</td>
<td>N/A</td>
<td>74024</td>
<td>1m45.560s</td>
<td>1m39.382s</td>
<td>37.69</td>
<td>4.898s</td>
<td>1.151</td>
</tr>
</tbody>
</table>

### Robot

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Explore Depth</th>
<th>States Generated</th>
<th>Total Time</th>
<th>User Time</th>
<th>Memory Used (MB)</th>
<th>Abstraction Time</th>
<th>Re-store Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>35865</td>
<td>0m12.838s</td>
<td>0m10.205s</td>
<td>35.70</td>
<td>0.0s</td>
<td>N/A</td>
</tr>
<tr>
<td>Static</td>
<td>N/A</td>
<td>27940</td>
<td>0m10.377s</td>
<td>0m8.229s</td>
<td>35.60</td>
<td>0.002s</td>
<td>N/A</td>
</tr>
<tr>
<td>Original best</td>
<td>2</td>
<td>27940</td>
<td>0m18.675s</td>
<td>0m16.641s</td>
<td>36.21</td>
<td>7.947s</td>
<td>N/A</td>
</tr>
<tr>
<td>Original worst</td>
<td>2</td>
<td>27940</td>
<td>0m18.675s</td>
<td>0m16.641s</td>
<td>36.21</td>
<td>7.947s</td>
<td>N/A</td>
</tr>
<tr>
<td>Maximal</td>
<td>N/A</td>
<td>27784</td>
<td>0m11.494s</td>
<td>0m09.525s</td>
<td>29.21</td>
<td>0.552s</td>
<td>1.28</td>
</tr>
</tbody>
</table>

### bintree

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Explore Depth</th>
<th>States Generated</th>
<th>Total Time</th>
<th>User Time</th>
<th>Memory Used (MB)</th>
<th>Abstraction Time</th>
<th>Re-store Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>N/A</td>
<td>157828</td>
<td>1m00.608s</td>
<td>0m49.811s</td>
<td>66.50</td>
<td>0.0s</td>
<td>N/A</td>
</tr>
<tr>
<td>Static</td>
<td>N/A</td>
<td>154084</td>
<td>1m01.061s</td>
<td>0m50.387s</td>
<td>68.40</td>
<td>0.005s</td>
<td>N/A</td>
</tr>
<tr>
<td>Original best</td>
<td>6</td>
<td>150964</td>
<td>2m14.807s</td>
<td>2m03.864s</td>
<td>73.74</td>
<td>72.09s</td>
<td>N/A</td>
</tr>
<tr>
<td>Original worst</td>
<td>2</td>
<td>154084</td>
<td>2m07.356s</td>
<td>1m56.635s</td>
<td>71.47</td>
<td>64.87s</td>
<td>N/A</td>
</tr>
<tr>
<td>Maximal</td>
<td>N/A</td>
<td>103839</td>
<td>1m07.530s</td>
<td>1m00.668s</td>
<td>52.62</td>
<td>16.34s</td>
<td>1.012</td>
</tr>
</tbody>
</table>
Conclusions

- Our algorithm generates DVA maximally reduced state spaces on-the-fly
- Uses less memory
- Requires no user specified depth bound
- Does well on models with loops
- Takes more time on some models
- Due to chained hash table and contains relation
Future Work

- Modify to work on multi-procedural programs
- Remove need for chained hash table and contains relation
- Adapt to other search algorithms
- Other static analysis techniques for precise abstraction?
Questions?
Maximum state space reduction from a DVA

For every reachable trace in the concrete state space there exists an abstract trace such that the states $s_i$ in the concrete trace are contained in states $s_i'$ in the abstract trace and $s_i'$ is in the abstract state space.

For all live variables in all states in the abstract state space, there exists a state in a reachable trace in the concrete state space where that variable is live in that state.
Static Vs. Dynamic

- Static dead variable analysis (SDVA) helps
- SDVA does not use dynamic run-time information
  - Variable valuations
  - Considers infeasible paths

- Dynamic Dead Variable Analysis (DDVA)
  - Uses variable valuation info
  - Only considers feasible paths
  - Finds more dead variables