CS 243
Lecture 11
Binary Decision Diagrams (BDDs) in Pointer Analysis

1. Relations in BDDs
2. Datalog -> Relational Algebra
3. Relational Algebra -> BDDs
4. Context-Sensitive Pointer Analysis
5. Performance of BDD Algorithms
6. Experimental Results

Readings: Chapter 12

Advanced Compilers

M. Lam & J. Whaley
L11. BDDs
Automatic Analysis Generation

Programmer: Security analysis in 10 lines

Compiler writer: Ptr analysis in 10 lines

PQL

Datalog

bddbdddb (BDD-based deductive database) with Active Machine Learning

BDD operations

BDD: 10,000s-lines library

1000s of lines 1 year tuning

1000s of lines library

BDD: 10,000s-lines library

PQL
1. Relations $\rightarrow$ BDDs

- Example

- calls(A,B)
- calls(A,C)
- calls(A,D)
- calls(B,D)
- calls(C,D)
Call Graph Relation

- Relation expressed as a binary function.
  - A=00, B=01, C=10, D=11
Binary Decision Diagrams (Bryant, 1986)

- Graphical encoding of a truth table.
Binary Decision Diagrams

- Collapse redundant nodes.
Collapse redundant nodes.
Binary Decision Diagrams

- Collapse redundant nodes.
Binary Decision Diagrams

- Collapse redundant nodes.
Eliminate unnecessary nodes.
Binary Decision Diagrams

- Eliminate unnecessary nodes.
2. Datalog to Relational Algebra

\[ t_1 = \rho_{\text{variable} \rightarrow \text{source}}(vP); \]
\[ t_2 = \text{assign} \bowtie t_1; \]
\[ t_3 = \pi_{\text{source}}(t_2); \]
\[ t_4 = \rho_{\text{dest} \rightarrow \text{variable}}(t_3); \]
\[ vP = vP \cup t_4; \]

**EXAMPLE**
\[ vP(\text{variable}, \text{obj}) \]
\[ \text{Assign(dest, source)} \]
\[ vP(v_1, o) :\text{- assign}(v_1, v_2), vP(v_2, o). \]
Semi-Naïve Evaluation

- Relations keep growing with each iteration
- The same computation is repeated with increasingly large tables – lot of redundant work
- Example
  
  \[ C(x,z) \leftarrow A(x,y), B(y,z) \]

  Let \( A_i, B_i, C_i \) be the value in iteration \( i \);
  
  \( \Delta \) be the diff with previous iteration.

  \[
  \begin{align*}
  C_i(x,z) & \leftarrow C_{i-1}(x,z) \\
  C_i(x,z) & \leftarrow \Delta A_{i-1}(x,y), B_{i-1}(y,z) \\
  C_i(x,z) & \leftarrow A_{i-1}(x,y), \Delta B_{i-1}(y,z)
  \end{align*}
  \]
**Example**

\( vP, \text{assign}: \) current values
\( vP’, \text{assign’}: \) old values
\( vP^{''}, \text{assign’’}: \) delta values

\[
\begin{align*}
    t_1 &= \rho_{\text{variable} \rightarrow \text{source}}(vP); \\
    t_2 &= \text{assign} \Join t_1; \\
    t_3 &= \pi_{\text{source}}(t_2); \\
    t_4 &= \rho_{\text{dest} \rightarrow \text{variable}}(t_3); \\
    vP &= vP \cup t_4;
\end{align*}
\]

\[
\begin{align*}
    vP^{''} &= vP - vP’; \\
    vP’ &= vP; \\
    \text{assign’’} &= \text{assign} - \text{assign’}; \\
    \text{assign’} &= \text{assign}; \\
    t_1 &= \rho_{\text{variable} \rightarrow \text{source}}(vP’’); \\
    t_2 &= \text{assign} \Join t_1; \\
    t_5 &= \rho_{\text{variable} \rightarrow \text{source}}(vP); \\
    t_6 &= \text{assign’’} \Join t_5; \\
    t_7 &= t_2 \cup t_6; \\
    t_3 &= \pi_{\text{source}}(t_7); \\
    t_4 &= \rho_{\text{dest} \rightarrow \text{variable}}(t_3); \\
    vP &= vP \cup t_4;
\end{align*}
\]
Eliminate Loop Invariant Combinations

\[
\begin{align*}
\nu P'' & = \nu P - \nu P'; \\
\nu P' & = \nu P; \\
\text{assign}'' & = \text{assign} - \text{assign}'; \\
\text{assign}' & = \text{assign}; \\
t_1 & = \rho_{\text{variable}\rightarrow\text{source}}(\nu P''); \\
t_2 & = \text{assign} \bowtie t_1; \\
t_5 & = \rho_{\text{variable}\rightarrow\text{source}}(\nu P); \\
t_6 & = \text{assign}'' \bowtie t_5; \\
t_7 & = t_2 \cup t_6; \\
t_3 & = \pi_{\text{source}}(t_7); \\
t_4 & = \rho_{\text{dest}\rightarrow\text{variable}}(t_3); \\
\nu P & = \nu P \cup t_4;
\end{align*}
\]

\[
\begin{align*}
\nu P'' & = \nu P - \nu P'; \\
\nu P' & = \nu P; \\
t_1 & = \rho_{\text{variable}\rightarrow\text{source}}(\nu P''); \\
t_2 & = \text{assign} \bowtie t_1; \\
t_3 & = \pi_{\text{source}}(t_2); \\
t_4 & = \rho_{\text{dest}\rightarrow\text{variable}}(t_3); \\
\nu P & = \nu P \cup t_4;
\end{align*}
\]

NOTE: assign never changes
### 3. Datalog $\rightarrow$ BDDs

<table>
<thead>
<tr>
<th>Datalog</th>
<th>BDDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relations</td>
<td>Boolean functions</td>
</tr>
<tr>
<td>Relation algebra:</td>
<td>Boolean function ops:</td>
</tr>
<tr>
<td>$\bowtie$, $\cup$, select, project</td>
<td>$\land$, $\lor$, $\neg$, $\sim$</td>
</tr>
<tr>
<td>Relation at a time</td>
<td>Function at a time</td>
</tr>
<tr>
<td>Semi-naïve evaluation</td>
<td>Incrementalization</td>
</tr>
<tr>
<td>Fixed-point</td>
<td>Iterate until stable</td>
</tr>
</tbody>
</table>
BDD: Relational Product (relprod)

- Relprod is a Quantified Boolean Formula
  \[ h = \exists x_1, x_2, \ldots f(x_1, x_2, \ldots) \land g(x_1, x_2, \ldots) \]

- \( h(v_1, \ldots v_n) \) is true if
  \[ \exists x_1, x_2, \ldots f(x_1, x_2, \ldots, v_i, \ldots) \land g(x_1, x_2, \ldots, v_j, \ldots) \]

- Same as an \( \land \) operation followed by projecting away common attributes

- Important because it is common and much faster to combine the operations in BDDs
Relational algebra -> BDD operations

\[
\begin{align*}
\text{relprod}: \text{relational product} \\
\text{diff}: \text{difference}
\end{align*}
\]

\[
\begin{align*}
vP'' &= vP - vP' \\
vP' &= vP \\
t_1 &= \rho_{\text{variable} \rightarrow \text{source}}(vP'') \\
t_2 &= \text{assign} \Join t_1 \\
t_3 &= \pi_{\text{source}}(t_2) \\
t_4 &= \rho_{\text{dest} \rightarrow \text{variable}}(t_3) \\
vP &= vP \cup t_4
\end{align*}
\]

NOTE: assign never changes
4. Context-Sensitive Pointer Analysis Algorithm

1. First, do context-insensitive pointer analysis to get call graph.
2. Number clones.
3. Do context-insensitive algorithm on the cloned graph.
   - Results explicitly generated for every clone.
   - Individual results retrievable with Datalog query.
Size of BDDs

- Represent tiny and huge relations compactly
- Size depends on redundancy
  - Similar contexts have similar numberings
  - Variable ordering in BDDs
BDD Variable Order is Important!

\[ x_1x_2 + x_3x_4 \]
Expanded Call Graph

Advanced Compilers

L11. BDDs
Numbering Clones

Advanced Compilers

L11. BDDs
5. Performance of Context-Sensitive Pointer Analysis

- Direct implementation
  - Does not finish even for small programs
  - > 3000 lines of code

- Requires tuning for about 1 year

- Easy to make mistakes
  - Mistakes found months later
An Adventure in BDDs

- **Context-sensitive numbering scheme**
  - Modify BDD library to add special operations.
  - Can’t even analyze small programs.  
    \[ \text{Time: } \infty \]

- **Improved variable ordering**
  - Group similar BDD variables together.
  - Interleave equivalence relations.
  - Move common subsets to edges of variable order.  
    \[ \text{Time: } 40h \]

- **Incrementalize outermost loop**
  - Very tricky, many bugs.  
    \[ \text{Time: } 36h \]

- **Factor away control flow, assignments**
  - Reduces number of variables  
    \[ \text{Time: } 32h \]
An Adventure in BDDs

- Exhaustive search for best BDD order
  - Limit search space by not considering intradomain orderings.  \(\text{Time: 10h}\)

- Eliminate expensive rename operations
  - When rename changes relative order, result is not isomorphic.  \(\text{Time: 7h}\)

- Improved BDD memory layout
  - Preallocate to guarantee contiguous.  \(\text{Time: 6h}\)

- BDD operation cache tuning
  - Too small: redo work, too big: bad locality
  - Parameter sweep to find best values.  \(\text{Time: 2h}\)
An Adventure in BDDs

- Simplified treatment of exceptions
  - Reduce number of vars, iterations necessary for convergence. \( \text{Time: 1h} \)

- Change iteration order
  - Required redoing much of the code. \( \text{Time: 48m} \)

- Eliminate redundant operations
  - Introduced subtle bugs. \( \text{Time: 45m} \)

- Specialized caches for different operations
  - Different caches for and, or, etc. \( \text{Time: 41m} \)
An Adventure in BDDs

- Compacted BDD nodes
  - 20 bytes → 16 bytes
  - Time: 38m

- Improved BDD hashing function
  - Simpler hash function.
  - Time: 37m

- Total development time: 1 year
  - 1 year per analysis?!?

- Optimizations obscured the algorithm.

- Many bugs discovered, maybe still more.

- Create bddbddb to make optimization available to all analysis writers using Datalog
Variable Numbering: Active Machine Learning

- Must be determined dynamically
- Limit trials with properties of relations
- Each trial may take a long time
- Active learning:
  - select trials based on uncertainty
- Several hours
- Comparable to exhaustive for small apps
Summary: Optimizations in bddbddb

- Algorithmic
  - Clever context numbering to exploit similarities

- Query optimizations
  - Magic-set transformation
  - Semi-naïve evaluation
  - Reduce number of rename operations

- Compiler optimizations
  - Redundancy elimination, liveness analysis, dead code elimination, constant propagation, definition-use chaining, global value numbering, copy propagation

- BDD optimizations
  - Active machine learning

- BDD library extensions and tuning
6. Experimental Results

- Top 20 Java projects on SourceForge
  - Real programs with 100K+ users each
- Using automatic bddbdddb solver
  - Each analysis only a few lines of code
  - Easy to try new algorithms, new queries
- Test system:
  - Pentium 4 2.2GHz, 1GB RAM
  - RedHat Fedora Core 1, JDK 1.4.2_04, javabdd library, Joeq compiler
Analysis time

$y = 0.0078x^{2.3233}$

$R^2 = 0.9197$
Analysis memory

\[ y = 0.3609x^{1.4204} \]

\[ R^2 = 0.8859 \]
Benchmark

Nine large, widely used applications

- Blogging/bulletin board applications
- Used at a variety of sites
- Open-source Java J2EE apps
- Available from SourceForge.net
## Vulnerabilities Found

<table>
<thead>
<tr>
<th>Category</th>
<th>SQL injection</th>
<th>HTTP splitting</th>
<th>Cross-site scripting</th>
<th>Path traversal</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Parameter</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>Cookie</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Non-Web</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>9</strong></td>
<td><strong>11</strong></td>
<td><strong>4</strong></td>
<td><strong>5</strong></td>
<td><strong>29</strong></td>
</tr>
</tbody>
</table>
# Accuracy

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Classes</th>
<th>Context insensitive</th>
<th>Context sensitive</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>jboard</td>
<td>264</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>blueblog</td>
<td>306</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>webgoat</td>
<td>349</td>
<td>51</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>blojsom</td>
<td>428</td>
<td>48</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>personalblog</td>
<td>611</td>
<td>460</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>snipsnap</td>
<td>653</td>
<td>732</td>
<td>27</td>
<td>12</td>
</tr>
<tr>
<td>road2hibernate</td>
<td>867</td>
<td>18</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>pebble</td>
<td>889</td>
<td>427</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>roller</td>
<td>989</td>
<td>378</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>5356</strong></td>
<td><strong>2115</strong></td>
<td><strong>41</strong></td>
<td><strong>12</strong></td>
</tr>
</tbody>
</table>

Advanced Compilers
Automatic Analysis Generation

Programmer: Security analysis in 10 lines

Compiler writer: Ptr analysis in 10 lines

PQL

Datalog

bdddbdddb (BDD-based deductive database) with Active Machine Learning

BDD operations

BDD: 10,000s-lines library

1000s of lines
1 year tuning

BDD: 10,000s-lines library

Compiler writer: Ptr analysis in 10 lines

Programmer: Security analysis in 10 lines

PQL

Datalog

bdddbdddb (BDD-based deductive database) with Active Machine Learning

BDD operations

BDD: 10,000s-lines library

1000s of lines
1 year tuning

BDD: 10,000s-lines library
Software

- System is publicly available at:
  http://bddbddd.sourceforge.net