CS 243
Lecture 13
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

Interprocedural Pointer Analysis

Object creation
pts(v, h) :- “h: T v = new T()”.

Assignment
pts(v₁, h₁) :- “v₁ = v₂” & pts(v₂, h₁).

Store
hpts(h₁, f, h₂) :- “v₁.f = v₂” & pts(v₁, h₁) & pts(v₂, h₂).

Load
pts(v₂, h₂) :- “v₂ = v₁.f” & pts(v₁, h₁) & hpts(h₁, f, h₂).

Parameter passing with virtual methods
invokes (s, m) :- “s: v.n (…)” & pts (v,h) & hType (h,t) & cha (t,n,m)
pts(v, h) :- invokes (s, m) &
            formal (m, i, v) & actual (s, i, w) & pts (w, h)
Cloning-Based Algorithm

- Apply the context-insensitive algorithm to the program to discover the call graph
- Find strongly connected components
- Create a “clone” for every context
- Apply the context-insensitive algorithm to cloned call graph

Whaley&Lam, PLDI 2004 (best paper award)

Time & Space

- Repeated execution
  - A small number of rules
  - Very large data set
    - \(10^{14}\) contexts
    - 1 byte for each context: 4 terabytes
Automatic Analysis Generation

Programmer:
Security analysis
in 10 lines

Compiler writer:
Ptr analysis
in 10 lines

1000s of lines
1 year tuning

PQL

Datalog

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

BDD operations

BDD: 10,000s-lines library

Outline

Binary Decision Diagrams (BDDs)
in Pointer Analysis

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

- Example

\[
\begin{array}{cccc}
A & B & C & D \\
\downarrow & \downarrow & \downarrow & \downarrow \\
\text{calls}(A,B) & \text{calls}(A,C) & \text{calls}(A,D) & \text{calls}(B,D) \text{ and calls}(C,D)
\end{array}
\]

Call Graph Relation

- Relation expressed as a binary function.
  - A=00, B=01, C=10, D=11

\[
\begin{array}{cccc|c}
x_1 & x_2 & x_3 & x_4 & f \\
\hline
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 0
\end{array}
\]
Binary Decision Diagrams (Bryant, 1986)

- Graphical encoding of a truth table.

### Binary Decision Diagrams

- Collapse redundant nodes.

---

Diagram of a binary decision diagram with nodes labeled $x_1, x_2, x_3, x_4$ and edges indicating 0 and 1 transitions. The diagram includes a truth table at the bottom with inputs $0111000100010000$. The diagram illustrates how redundant nodes are collapsed.
Binary Decision Diagrams

- Collapse redundant nodes.

Binary Decision Diagrams

- Collapse redundant nodes.
Binary Decision Diagrams

- Collapse redundant nodes.

Binary Decision Diagrams

- Eliminate unnecessary nodes.
Binary Decision Diagrams

- Eliminate unnecessary nodes.

What’s the size of

- An empty set?
- The Universal set?
**BDD Variable Order is Important to the size!**

$x_1x_2 + x_3x_4$

- $x_3, x_4$ in the left diagram lead to the same 1, while in the right diagram, they lead to different 0 and 1.

**Reduced Ordered BDD**

- **Ordered**
  - Variables are in a fixed order

- **Reduced**
  - Nodes are reduced to create a compact representation

- The ROBDD representation of a binary function is unique
BDD Operations

- **apply** \((\text{op}, B_1, B_2)\)
  - 16 2-input logical functions
- **restrict** \((c, x, B)\)
  - Restrict variable \(x\) to constant \(c = 0\) or \(1\)
- **exists** \((x, B)\)
  - Does there exist \(x\) such that \(B\) is true?

Apply

- \(B = \text{apply} (\text{op}, B_1, B_2)\)
  - Combine two binary functions with a logical operator
  - \(B\) is a BDD that provides the answers to all possible inputs for \(B_1 \text{ op } B_2\)
16 2-input Boolean Operators

<table>
<thead>
<tr>
<th>X</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>False</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X and Y</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X &gt; Y</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X &lt; Y</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X XOR Y</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X OR Y</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X NOR Y</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NOT Y</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>X ≥ Y</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NOT X</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X ≤ Y</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X NAND Y</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>True</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Algorithm: Apply

\[
\text{Apply}(\text{op}, \text{B}, \text{C}) = \begin{cases} 
\text{Apply}(\text{op}, \text{B}, \text{C}) \text{ Apply}(\text{op}, \text{B}', \text{C}') 
\end{cases}
\]

Where \(C\) is (1) a terminal node or (2) a non-terminal with \(\text{var}(\text{root}(C)) > x\)

\[
\text{Apply}(\text{op}, \text{B}, \text{C}) = \begin{cases} 
\text{Apply}(\text{op}, \text{B}, \text{C}) \text{ Apply}(\text{op}, \text{B}', \text{C}') 
\end{cases}
\]

Where \(B\) is (1) a terminal node or (2) a non-terminal with \(\text{var}(\text{root}(B)) > x\)

\[
\text{Apply}(\text{op}, \text{u}, \text{v}) = w, \text{ where } w = u \text{ op } v
\]
Example: Apply

Example: Apply (OR, R, S)
Example: Apply (OR, R, S)
Example: Apply (OR, R, S)

Algorithm: Restrict

- restrict(c, x, B)
  - Restrict variable x to constant c = 0 or 1
Algorithm: Exists

- $B_1 = \text{exists}(x, B)$
  
  $= \text{apply (OR, restrict (0,x,B), restrict (1,x,B))}$

- $B_1 = 0$ if there does not exist an $x$
  
  $= \text{binary function (without variable } x\text{)}$

  that defines when there exists an $x$

  such that $B$ is true.

---

Does there exist $x_1$ such that $B$ is true?

\[(x_1 \land x_2) \lor (\overline{x_1} \land x_3)\]

\(x_3\)

\(x_2\)

\(x_1\)

\(0\)

\(1\)

\(\text{When?}\)

restrict(0, $x_1$, B)

restrict(1, $x_1$, B)

restrict(0, $x_1$, B) OR restrict(1, $x_1$, B)

\(x_3\lor x_3\)

\(x_2\)

\(x_1\)
BDD: Relational Product (relprod)

- Relprod is a Quantified Boolean Formula
- \( h = \text{Relprod}(f, g, [x_1, x_2, \ldots]) \)
  - \( h(v_1, \ldots, v_n) \) 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 \( x_1, x_2, \ldots \)
- Important because it is common and much faster to combine the \( \land \) and projection operations in BDDs

Outline

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
2. Datalog to Relational Algebra

- Relational Algebra
  - A theoretic foundation for relational databases
  - E.g. SQL

Five Relational Algebra Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \cup )</td>
<td>Set Union</td>
</tr>
<tr>
<td>(-)</td>
<td>Set Difference</td>
</tr>
<tr>
<td>( \rho_{\text{old} \rightarrow \text{new}} )</td>
<td>Rename old with name</td>
</tr>
<tr>
<td>( \pi_c )</td>
<td>Project away column c</td>
</tr>
<tr>
<td>( \bowtie )</td>
<td>Join two relations based on common column name</td>
</tr>
</tbody>
</table>

**EXAMPLE**

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

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

- Translate recursion into a Repeat loop
- Let S be the state of the computation

Do

\[ S' = S; \]
\[ S = \text{Apply-a-rule (} S'\text{);} \]

Until \( S = S' \)

Optimization:
Semi-Naïve Evaluation

- Relations keep growing with each iteration
- The same computation is repeated with increasingly large tables – lots of redundant work

Example

\[ C(x,z) : - A(x,y), B(y,z) \]

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

\[ C_i(x,z) : - C_{i-1}(x,z) \]
\[ C_i(x,z) : - \Delta A_{i-1} (x,y), B_{i-1} (y,z) \]
\[ C_i(x,z) : - A_{i-1} (x,y), \Delta B_{i-1} (y,z) \]
Example

\( vP(v_1, o) \) :: \( \text{assign}(v_1, v_2), vP(v_2, o) \).

\( vP \), assign: current values
\( vP' \), assign': old values
\( vP'' \), assign'': delta values

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

**NOTE:** assign never changes

Eliminate Loop Invariant Computations

\( vP'' = vP - vP'; \)
\( vP' = vP; \)
\( \text{assign''} = \text{assign} - \text{assign'}; \)
\( \text{assign'} = \text{assign}; \)
\[ t_1 = \rho_{\text{variable-source}}(vP''); \]
\[ t_2 = \text{assign} \bowtie t_1; \]
\[ t_3 = \pi_{\text{source}}(t_2); \]
\[ t_4 = \rho_{\text{dest-variable}}(t_3); \]
\[ t_5 = \rho_{\text{variable-source}}(vP); \]
\[ t_6 = \text{assign''} \bowtie t_5; \]
\[ t_7 = t_2 \cup t_6; \]
\[ vP = vP \cup t_4; \]

**NOTE:** assign never changes
# Outline

**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

---

## 3. Datalog → 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: (\bowtie), (\cup), select, project</td>
<td>Boolean function ops: apply, restrict, exists, relprod</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>
Relational algebra ->
BDD operations

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

\[
\begin{align*}
vP'' &= \text{diff}(vP, vP'); \\
vP' &= \text{copy}(vP); \\
t_1 &= \text{replace}(vP'', \text{variable} \rightarrow \text{source}); \\
t_3 &= \text{relprod}(t_1, \text{assign}, \text{source}); \\
t_4 &= \text{replace}(t_3, \text{dest} \rightarrow \text{variable}); \\
vP &= \text{or}(vP, t_4);
\end{align*}
\]

NOTE: assign never changes

Outline
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
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
Expanded Call Graph

Numbering Clones
Outline

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

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 \( \text{Time}: 32h \)

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. \( \text{Time: 41m} \)

- Compacted BDD nodes
  - 20 bytes \( \rightarrow \) 16 bytes \( \text{Time: 38m} \)

- Improved BDD hashing function
  - Simpler hash function. \( \text{Time: 37m} \)

- 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 bddbddd

- 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
Outline

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

6. Experimental Results

- Top 20 Java projects on SourceForge
  - Real programs with 100K+ users each
- Using automatic bddbddb 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></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>Total</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>Total</td>
<td>5356</td>
<td>2115</td>
<td>41</td>
<td>12</td>
</tr>
</tbody>
</table>

### Automatic Analysis Generation

- **Programmer:** Security analysis in 10 lines
- **Compiler writer:** Ptr analysis in 10 lines
- **PQL**
- **Datalog**
- **bddbddd (BDD-based deductive database)** with Active Machine Learning
- **BDD operations**
- **BDD: 10,000s-lines library**
- 1000s of lines
- 1 year tuning
Software

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