CS 243
Lecture 13
Binary Decision Diagrams (BDDs)
in Pointer Analysis

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

Readings: Chapter 12

Automatic Analysis Generation

Programmer: Security analysis in 10 lines

Compiler writer: Ptr analysis in 10 lines

1000s of lines 1 year tuning

PQL

Datalog

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

BDD operations

BDD: 10,000s-lines library
### Interprocedural Pointer Analysis

**Object creation**
\[ \text{pts}(v, h) : \text{“}h : T \ v = \text{new} \ T() \text{“}. \]

**Assignment**
\[ \text{pts}(v_1, h_1) : \text{“}v_1 = v_2 \text{“} \& \text{pts}(v_2, h_1). \]

**Store**
\[ \text{hpts}(h_1, f, h_2) : \text{“}v_1.f = v_2 \text{“} \& \text{pts}(v_1, h_1) \& \text{pts}(v_2, h_2). \]

**Load**
\[ \text{pts}(v_2, h_2) : \text{“}v_2 = v_1.f \text{“} \& \text{pts}(v_1, h_1) \& \text{hpts}(h_1, f, h_2). \]

**Parameter passing with virtual methods**
\[ \text{invokes} \ (s, m) : \text{“}s : v.n (...) \text{“} \& \text{pts}(v, h) \& \text{hType}(h, t) \& \text{cha}(t, n, m) \]
\[ \text{pts}(v, h) : \text{“} \text{invokes} \ (s, m) \& \text{formal} (m, i, v) \& \text{actual} (s, i, w) \& \text{pts}(w, h) \text{“}. \]

### 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

Outline

Binary Decision Diagrams (BDDs) in Pointer Analysis

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

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

Five Relational Algebra Operators

- \( \cup \) Set Union
- \( \neg \) Set Difference
- \( \rho_{old \rightarrow new} \) Rename old with name
- \( \pi_c \) Project away column \( c \)
- \( \bowtie \) Join two relations based on common column name

EXAMPLE

\[
\begin{align*}
\text{vP} & \text{variable, obj) } \\
\text{Assign(dest, source) } & \\
\text{vP}(v_1, o) & : \text{assign}(v_1, v_2), \text{vP}(v_2, o).
\end{align*}
\]

\[
\begin{align*}
\text{t}_1 & = \rho_{\text{variable} \rightarrow \text{source}}(\text{vP}); \\
\text{t}_2 & = \text{assign} \bowtie \text{t}_1; \\
\text{t}_3 & = \pi_{\text{source}}(\text{t}_2); \\
\text{t}_4 & = \rho_{\text{dest} \rightarrow \text{variable}}(\text{t}_3); \\
\text{vP} & = \text{vP} \cup \text{t}_4;
\end{align*}
\]
Translating Datalog to Relational Algebra

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

Do

S' = S;
S = Apply-a-rule (S');

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 \( i \); let \( \Delta \) 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

\[ vP(v_1, o) \] \::= \: assign(v_1, v_2), vP(v_2, o). \]

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

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

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

Eliminate Loop Invariant Computations

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

NOTE: assign never changes
Outline
Binary Decision Diagrams (BDDs) in Pointer Analysis

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

2. Introduction to BDDs

- BDD: Binary Decision Diagrams
- Designed to exploit similarities in an exponential number of states
Relations as BDDs

- Example

```
A
/\  
B  C
/ \  /
D  
```
calls(A,B)
calls(A,C)
calls(A,D)
calls(B,D)
calls(C,D)

Call Graph Relation

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- 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.

```
0 1 1 1 1 0 0 0 1 0 0 1 0 0 0 0
```

Binary Decision Diagrams

- Collapse redundant nodes.

```
0 1 1 1 0 0 0 1 0 0 1 0 0 0 0 0
```
Binary Decision Diagrams

- Collapse redundant nodes.

![Diagram of binary decision diagrams](image)
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!

\[ x_1 x_2 + x_3 x_4 \]

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
Outline

Binary Decision Diagrams (BDDs) in Pointer Analysis

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

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>\cup, select, project, \bowtie</td>
<td>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>
Basic 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></th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<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>1</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

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

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

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

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

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

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

\[(x_1 \land x_2) \lor x_4 \lor (x_2 \land x_3)\]  
\[(x_1 \land x_3) \lor x_4\]

Algorithm: Restrict

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

 restrict(0, x_3, B)
Algorithm: Exists

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

- $B_1 = 0$ if there does not exist an $x$
  = binary function (without variable $x$)
    that defines when there exists an $x$
    such that $B$ is true.

---

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

Resolve

$$
\begin{align*}
\text{Resolve} & \quad p \lor A \\
& = p \lor B \\
& = A \lor B
\end{align*}
$$
BDD: Relational Product (relprod)

- Relprod is a Quantified Boolean Formula (Corresponding to join + project in relational algebra)
- \( h = \text{Relprod}(f, g, [x_1, x_2, \ldots]) \)
  - \( h(v_1, \ldots, v_n) \) is true if
    \( \exists x_1, x_2, \ldots \cdot f(x_1, x_2, \ldots, v_i, \ldots) \land g(x_1, x_2, \ldots, v_j, \ldots) \)
- 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

Relational algebra -> BDD operations

\[
\begin{align*}
\text{vP}'' &= \text{vP} - \text{vP}'; \\
\text{vP}' &= \text{vP}; \\
\text{t}_1 &= \rho_{\text{variable} \to \text{source}}(\text{vP}'') \\
\text{t}_2 &= \text{assign} \land \text{t}_1; \\
\text{t}_3 &= \pi_{\text{source}}(\text{t}_2); \\
\text{t}_4 &= \rho_{\text{dest} \to \text{variable}}(\text{t}_3); \\
\text{vP} &= \text{vP} \cup \text{t}_4; \\
\text{vP}'' &= \text{diff}(\text{vP}, \text{vP}'); \\
\text{vP}' &= \text{copy}(\text{vP}); \\
\text{t}_1 &= \text{replace}(\text{vP}'', \text{variable} \to \text{source}); \\
\text{t}_3 &= \text{relprod}(\text{t}_1, \text{assign}, \text{source}); \\
\text{t}_4 &= \text{replace}(\text{t}_3, \text{dest} \to \text{variable}); \\
\text{vP} &= \text{or}(\text{vP}, \text{t}_4); \\
\end{align*}
\]

NOTE: assign never changes
Outline
Binary Decision Diagrams (BDDs) in Pointer Analysis

1. Datalog -> Relational Algebra
2. Relations in BDDs
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

```
A               A_0
  |              /\      /
  B              B_0  C_0  D_0
    \          /\      /
      E         E_0  E_1  E_2
        /\      /
       F       F_0  F_1  F_2
          /\      /
         G      G_0  G_1  G_2
            /\      /
           H      H_0  H_1  H_2  H_3  H_4  H_5
```

Advanced Compilers
M. Lam & J. Whaley
Numbering Clones

Outline
Binary Decision Diagrams (BDDs) in Pointer Analysis

1. Datalog -> Relational Algebra
2. Relations in BDDs
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 \( \text{Time: } 32h \)
An Adventure in BDDs

- Exhaustive search for best BDD order
  - Limit search space by not considering intradomain orderings.  \textit{Time: 10h}
- Eliminate expensive rename operations
  - When rename changes relative order, result is not isomorphic.  \textit{Time: 7h}
- Improved BDD memory layout
  - Preallocate to guarantee contiguous.  \textit{Time: 6h}
- BDD operation cache tuning
  - Too small: redo work, too big: bad locality
  - Parameter sweep to find best values.  \textit{Time: 2h}

An Adventure in BDDs

- Simplified treatment of exceptions
  - Reduce number of vars, iterations necessary for convergence.  \textit{Time: 1h}
- Change iteration order
  - Required redoing much of the code.  \textit{Time: 48m}
- Eliminate redundant operations
  - Introduced subtle bugs.  \textit{Time: 45m}
- Specialized caches for different operations
  - Different caches for and, or, etc.  \textit{Time: 41m}
An Adventure in BDDs

- Compacted BDD nodes
  - 20 bytes → 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 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

Outline

Binary Decision Diagrams (BDDs) in Pointer Analysis

1. Datalog -> Relational Algebra
2. Relations in BDDs
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 bddbddd 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

\[ y = 0.0078x^{2.3233} \]
\[ R^2 = 0.9197 \]
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><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>
Automatic Analysis Generation

Programmer:
Security analysis
in 10 lines

Compiler writer:
Ptr analysis
in 10 lines

PQL

Datalog

bddbddb
(BDD-based
deductive database)

with
Active Machine Learning

BDD: 10,000s-lines library

1000s of lines
1 year tuning

BDD operations

Software

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