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 Conservative Analysis Generation

Programmer: Security analysis in 10 lines

Compiler Writer: Ptr analysis in 10 lines

BDD (Binary Decision Diagrams): 10,000s-lines library

PQL

Datalog

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

BDD operations

1000s of lines
1 year tuning
Interprocedural Pointer Analysis

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

Assignment
\[ \text{pts}(v_1, h_1) : - "v_1 = v_2", \text{pts}(v_2, h_1). \]

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

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

Parameter passing with virtual methods
\[ \text{invokes}(s, m) : - "s: v.n(...)", \text{pts}(v,h), \text{hType}(h,t), \text{cha}(t,n,m). \]
\[ \text{pts}(v, h) : - \text{invokes}(s, m), \text{formal}(m, i, v), \]
\[ \text{actual}(s, i, w), \text{pts}(w, h). \]
Cloning-Based Algorithm

• Apply the context-insensitive algorithm to the program to discover the call graph

• Context-sensitive analysis
  – Find strongly connected components
  – Create a “clone” for every context
  – Apply the context-insensitive algorithm to cloned call graph
Behavior of the Program

- Computing 3 tables for the whole program:
  - $pts(v,h)$, $hpts(h_1,f,h_2)$, $invokes(s,m)$

- Giant tables:
  - Context-sensitivity: $10^{14}$ clones
    - 47 bits to number the clones
    - If we need just 1 byte for each context: 100 terabytes

- Applying 6 rules
  - Each application operates on entire tables

- The rules are applied repeatedly many times
  - The tables grow monotonically
  - Lots of repeated computation
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

- \( U \) Set Union
- \( - \) 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**

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

\[
\begin{align*}
  t_1 &= \rho_{\text{variable} \rightarrow \text{source}}(\text{vP}); \\
  t_2 &= \text{assign} \bowtie t_1; \quad // (v_1, v_2, o) \\
  t_3 &= \pi_{\text{source}}(t_2); \quad // (v_1, o) \\
  t_4 &= \rho_{\text{dest} \rightarrow \text{variable}}(t_3); \\
  \text{vP} &= \text{vP} \cup t_4;
\end{align*}
\]
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
• Semi-naïve evaluation: only compute the changed tuples

• Example

C(x,z) :- A(x,y), B(y,z).

Let $A_i$, $B_i$, $C_i$ be the value in iteration $i$;

$\Delta A_i$ be the diff between $A_i$, $A_{i-1}$
$\Delta B_i$ be the diff between $B_i$, $B_{i-1}$

$C_i(x,z) :- C_{i-1}(x,z)$.

$C_i(x,z) :- \Delta A_i(x,y), B_i(y,z)$.

$C_i(x,z) :- A_i(x,y), \Delta B_i(y,z)$. 
Example

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

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

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

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
- Usage: logic synthesis, verification
Relations as 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

\[ f(x_1, x_2, x_3, x_4) = \text{calls}(\langle x_1, x_2 \rangle, \langle x_3, x_4 \rangle) \]

<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>1</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>1</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>1</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>0</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>1</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>1</td>
</tr>
</tbody>
</table>

**Diagram:**

- A (00) → B (01)
- A (00) → C (10)
- B (01) → C (10)
- C (10) → D (11)
- D (11) → A (00)
- D (11) → B (01)
- D (11) → C (10)
Binary Decision Diagrams (Bryant, 1986)

- Graphical encoding of a truth table.
Binary Decision Diagrams

• Collapse redundant nodes.
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!

\[ x_1x_2 + x_3x_4 \]

\(x_1, x_2, x_3, x_4\) vs \(x_1, x_3, x_2, x_4\)
Reduced Ordered BDD

• Ordered
  – Variables are in a fixed order

• Reduced
  – Nodes are reduced to create a compact representation

• The ROBDD (Reduced, Ordered) 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 \( B_1 \text{ op } B_2 \)
## 2-input Boolean Operators: 16 Combinations

<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>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>0</td>
</tr>
<tr>
<td>X NOR Y</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X XNOR Y</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</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

Apply(op, \( B, B' \), \( C, C' \)) = Apply(op, B, C) Apply(op, B', C')

Apply(op, \( B, B' \), \( C \)) = Apply(op, B, C) Apply(op, B', C)

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

Apply(op, \( B \), \( C, C' \)) = Apply(op, B, C) Apply(op, B, C')

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

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

- Combine the BDDs for generic op

E1: \((x_1 \land x_3) \lor x_4 \lor (x_2 \land x_3)\)

E2: \((x_1 \land x_3) \lor x_4\)

Question: what is E1 \lor E2?
Example: Apply \((OR, R, S)\)

- Apply OR to the constant nodes

E1: \((x_1 \land x_3) \lor x_4 \lor (x_2 \land x_3)\)  E2: \((x_1 \land x_3) \lor x_4\)
Example: Apply (OR, R, S)

- Collapse redundant nodes

\[ E_1: (x_1 \land x_3) \lor x_4 \lor (x_2 \land x_3) \quad E_2: (x_1 \land x_3) \lor x_4 \]

\[ E_1 \lor E_2 \]
Example: Apply (OR, R, S)

- Collapse redundant nodes

E1: $(x_1 \land x_3) \lor x_4 \lor (x_2 \land x_3)$  E2: $(x_1 \land x_3) \lor x_4$

$E1 \lor E2$
Example: Apply (OR, R, S)

- Collapse redundant nodes

E1: \((x_1 \land x_3) \lor x_4 \lor (x_2 \land x_3)\)  
E2: \((x_1 \land x_3) \lor x_4\)
Example: Apply (OR, R, S)

E1: \((x_1 \land x_3) \lor x_4 \lor (x_2 \land x_3)\)  
E2: \((x_1 \land x_3) \lor x_4 \lor (x_2 \land x_3)\)  

E1 \lor E2
Algorithm: Restrict

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

restrict(0, x_3, B)
Algorithm: Exists

Does there exist $x_1$ such that $E$ is true?
When does there exist an $x_1$ such that $E$ is true?

Useful inference rule:
Resolve $p \lor A$  $\neg p \lor B$

\[
\begin{align*}
A \lor B
\end{align*}
\]

• $B_1 = \text{exists}(x,B)$
  = 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.

$E: (x_1 \land x_2) \lor (\overline{x_1} \land x_3)$
Does there exist $x_1$ such that B is true?

$$(x_1 \land x_2) \lor (\overline{x_1} \land x_3)$$
BDD: Relational Product (relprod)

• Relprod is a Quantified Boolean Formula
  (Corresponding to join + project in relational algebra)

• $h = \text{Relprod}(f, g, [x_1, x_2, ...])$
  
  
  h($v_1$, ... $v_n$) is true if

  $\exists x_1, x_2, ..., f(x_1, x_2, ..., v_i, ...) \land g(x_1, x_2, ..., v_j, ...)$

• Same as an $\land$ operation
  followed by projecting away common attributes $x_1, x_2, ...$

• Important because it is common and much faster to combine
  the $\land$ and projection operations in BDDs
Relational algebra -> BDD operations

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

\[ vP'' = \text{diff}(vP, vP'); \]
\[ vP' = \text{copy}(vP); \]
\[ t_1 = \text{replace}(vP'', \text{variable} \rightarrow \text{source}); \]
\[ t_3 = \text{relprod}(t_1, \text{assign}, \text{source}); \]
\[ t_4 = \text{replace}(t_3, \text{dest} \rightarrow \text{variable}); \]
\[ vP = \text{or}(vP, t_4); \]
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
Numbering Clones

![Diagram of cloning numbers]

- A
- B
- C
- D
- E
- F
- G
- H

- A0
- B0
- C0
- D0
- E0
- E1
- E2
- F0
- F1
- F2
- G0
- G1
- G2
- H0
- H1
- H2
- H3
- H4
- H5
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
Cloning-Based Algorithm

• Apply the context-insensitive algorithm to the program to discover the call graph

• Context-sensitive analysis
  – Find strongly connected components
  – Create a “clone” for every context
  – Apply the context-insensitive algorithm to cloned call graph
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.  
    *Time: 10h*

• Eliminate expensive rename operations
  – When rename changes relative order, result is not isomorphic.  
    *Time: 7h*

• Improved BDD memory layout
  – Preallocate to guarantee contiguous.  *Time: 6h*

• BDD operation cache tuning
  – Too small: redo work, too big: bad locality
  – Parameter sweep to find best values.  *Time: 2h*
An Adventure in BDDs

• Simplified treatment of exceptions
  – Reduce number of vars, iterations necessary for convergence. *Time: 1h*

• Change iteration order
  – Required redoing much of the code. *Time: 48m*

• Eliminate redundant operations
  – Introduced subtle bugs. *Time: 45m*

• Specialized caches for different operations
  – Different caches for and, or, etc. *Time: 41m*
An Adventure in BDDs

• Compacted BDD nodes
  – 20 bytes $\rightarrow$ 16 bytes
  
• Improved BDD hashing function
  – Simpler hash function.
  
• Total development time: 1 year
  – 1 year per analysis?!?

• Optimizations obscured the algorithm.
• Many bugs discovered, maybe still more.
• Create bddbddd 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 bddbd
d
• 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 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>9</td>
<td>11</td>
<td>4</td>
<td>5</td>
<td>29</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 Conservative 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

1000s of lines
1 year tuning

BDD operations

BDD (Binary Decision Diagrams): 10,000s-lines library
General Lessons

• BDD: A (magical) data structure for exponential amount of information
  – No free lunch: only if redundancy exists
  – Not suitable for random information
  – Not easy to ”tame” either

• Pointer alias analysis
  – Many “clever” attempts to exploit program semantics failed to scale
  – Imprecision causes the representation to explode

• Reuse of languages and libraries
  – Key software engineering productivity