CS 243

Lecture 14

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

Automatic Analysis Generation

Programmer:
Security analysis
in 10 lines

PQL

Compiler writer:
Ptr analysis
in 10 lines

Datalog

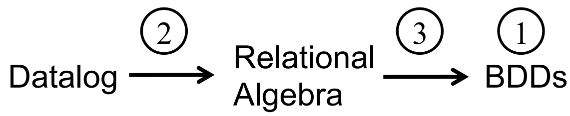
bddbddb
(**BDD-b**ased
deductive **d**ata**b**ase)
with
Active Machine Learning

1000s of lines
1 year tuning

BDD operations

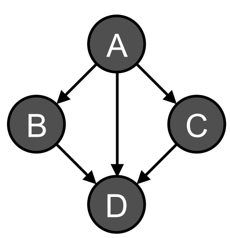
BDD: 10,000s-lines library

Outline



1. Relations → BDDs

- Example

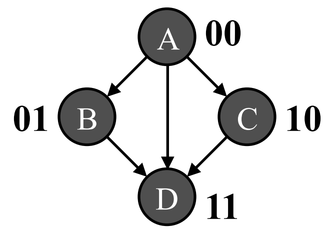


calls(A,B)
calls(A,C)
calls(A,D)
calls(B,D)
calls(C,D)

Call Graph Relation

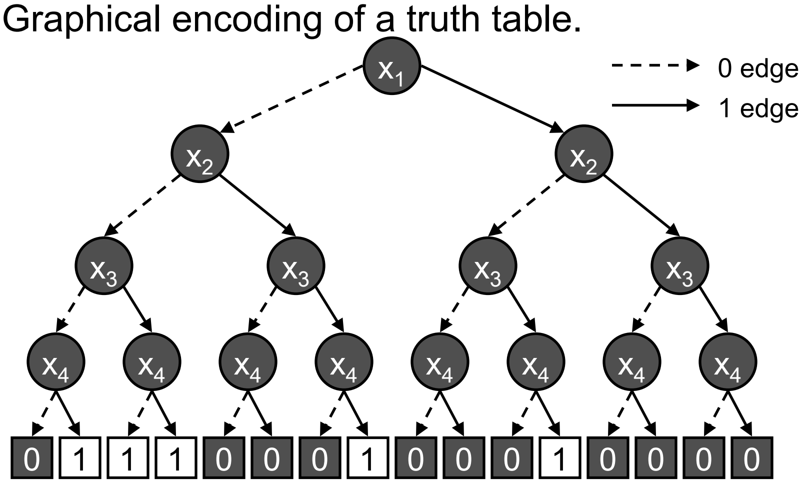
x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

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



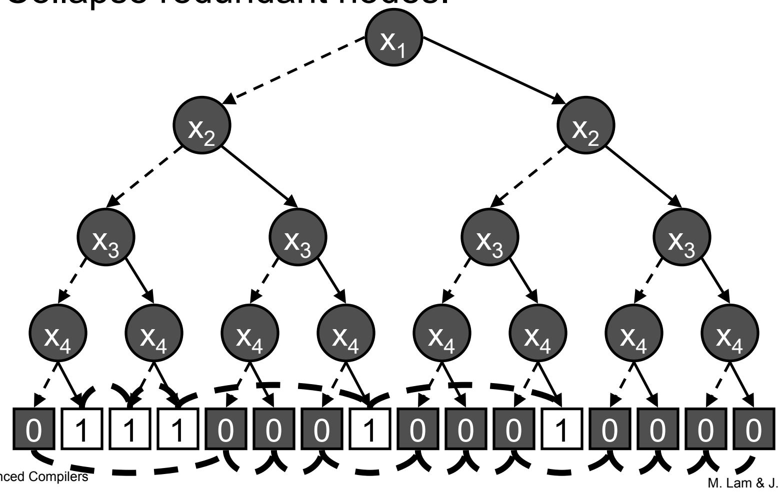
Binary Decision Diagrams (Bryant, 1986)

- Graphical encoding of a truth table.



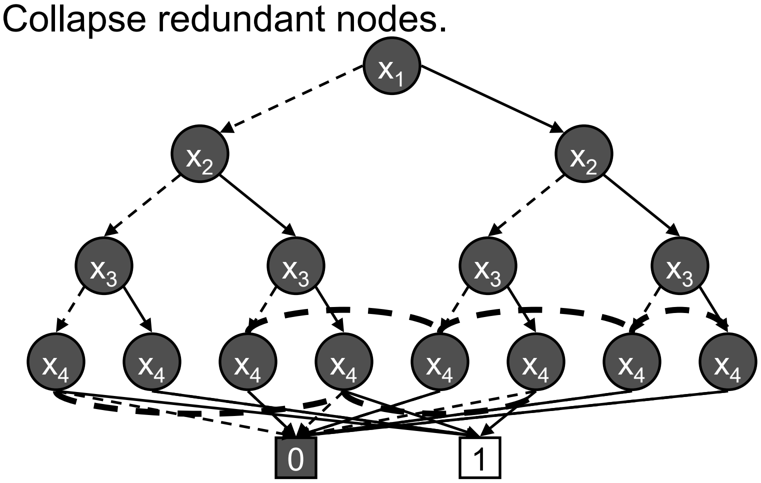
Binary Decision Diagrams

- Collapse redundant nodes.



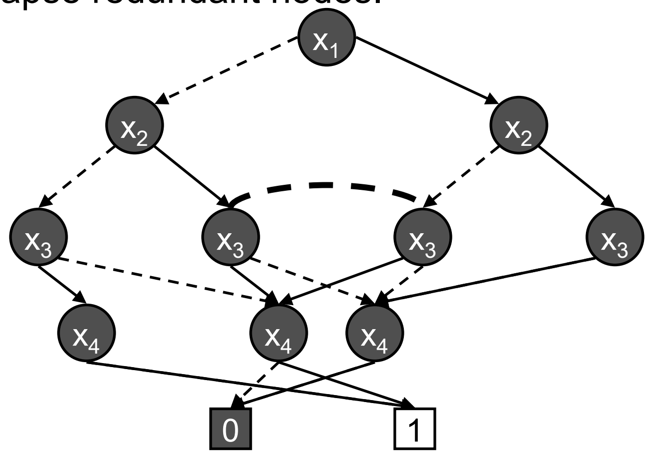
Binary Decision Diagrams

- Collapse redundant nodes.



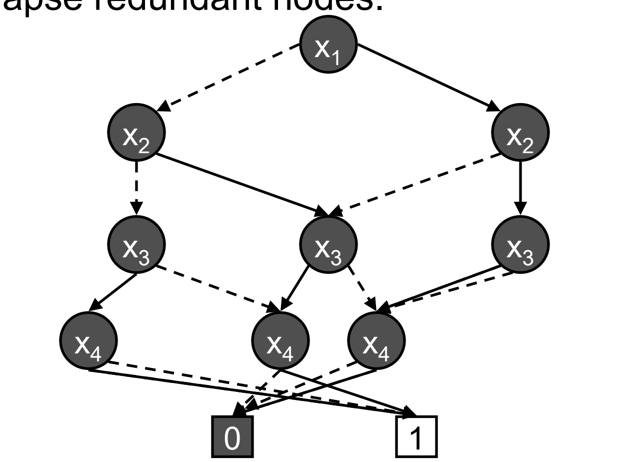
Binary Decision Diagrams

- Collapse redundant nodes.



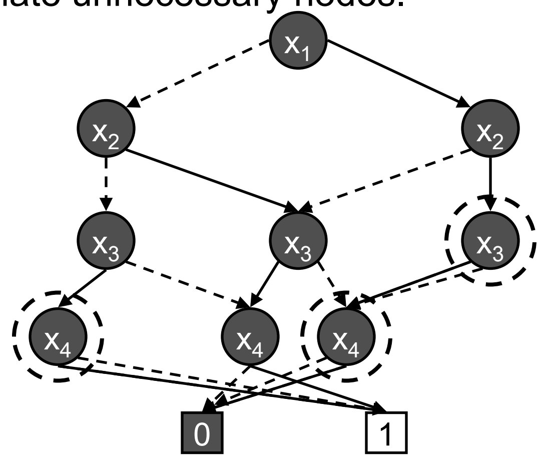
Binary Decision Diagrams

- Collapse redundant nodes.



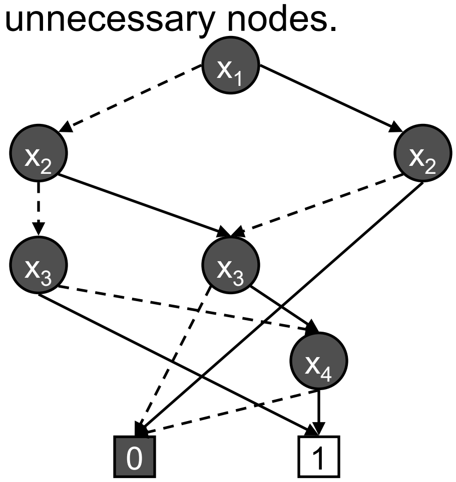
Binary Decision Diagrams

- Eliminate unnecessary nodes.



Binary Decision Diagrams

- Eliminate unnecessary nodes.



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 (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?

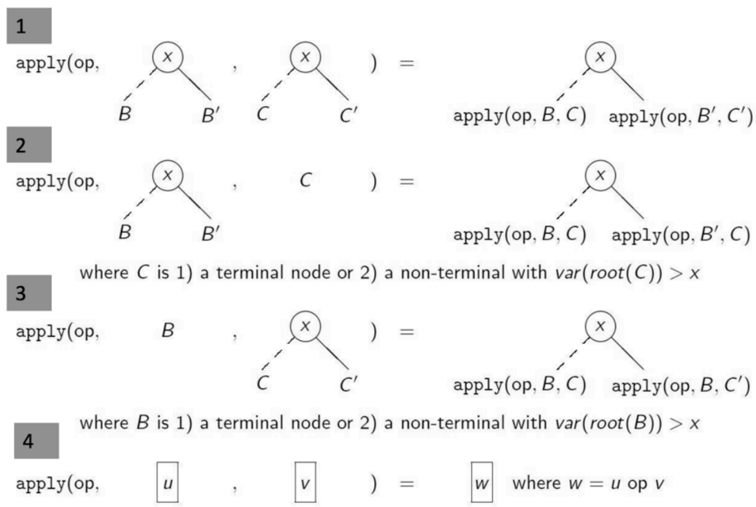
Apply

- B = apply (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 op B_2

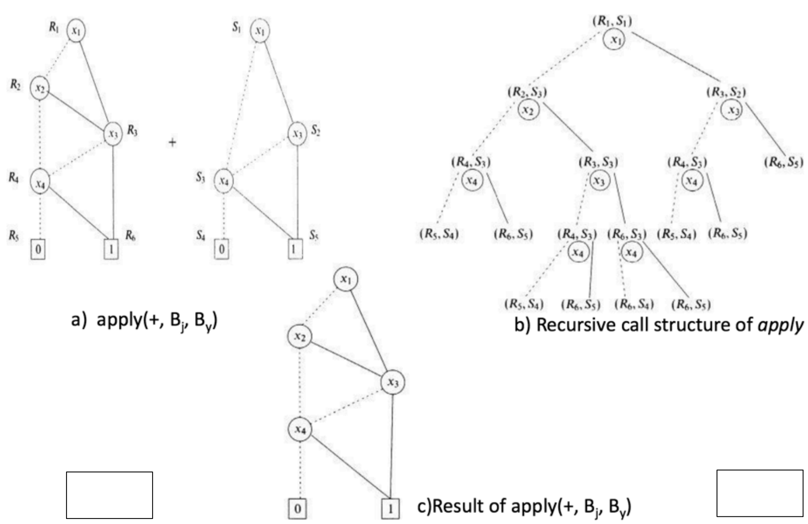
16 2-input Boolean Operators

X	0	0	1	1
Y	0	1	0	1
False	0	0	0	0
X and Y	0	0	0	1
X > Y	0	0	1	0
X	0	0	1	1
X < Y	0	1	0	0
Y	0	1	0	1
X XOR Y	0	1	1	0
X OR Y	0	1	1	1
X NOR Y	1	0	0	0
X XNOR Y	1	0	0	1
NOT Y	1	0	1	0
X ≥ Y	1	0	1	1
NOT X	1	1	0	0
X ≤ Y	1	1	0	1
X NAND Y	1	1	1	0
True	1	1	1	1

Algorithm: Apply



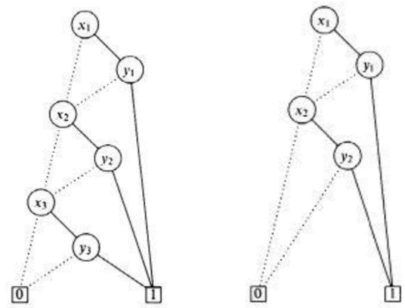
Example: Apply



Algorithm: Restrict

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

restrict(0, x_3, B)

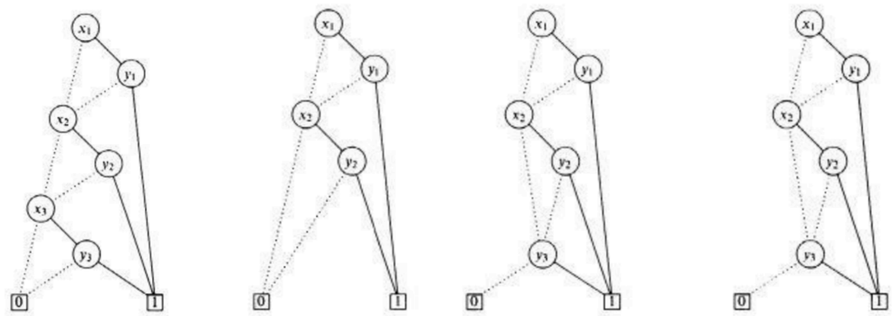


Haroon Rashid

Algorithm: Exists

- B_1 = exists(x,B)
  = apply (+, 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.

Example: Exists



B | restrict(0, x_3, B) | restrict(1, x_3, B) | restrict(0, x_3, B) + restrict(1, x_3, B)

Haroon Rashid

★When does there exist an x such that B is true?

Outline

Datalog —(2)→ Relational Algebra —(3)→ (1) BDDs

2. Datalog to Relational Algebra

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

Five Relational Algebra Operators

$\cup$	Set Union
-	Set Difference
$\rho_{old \rightarrow new}$	Rename old with name
π_c	Projection away column c
$\bowtie$	Join two relations based on common column name

EXAMPLE
vP(variable, obj)
Assign(dest, source)

$vP(v_1, o)$:- $assign(v_1, v_2), vP(v_2, o)$.

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

Translating Datalog to Relational Algebra

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

Do

S' = S;

S = 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 – lot 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;

  Δ be the diff with previous iteration.

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

  $C_i(x,z)$:- ΔA_{i-1} (x,y), B_{i-1} (y,z)

  $C_i(x,z)$:- A_{i-1} (x,y), ΔB_{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

$t_1 = \rho_{variable \to source}(vP)$;
$t_2 = assign \bowtie t_1$;
$t_3 = \pi_{source}(t_2)$;
$t_4 = \rho_{dest \to variable}(t_3)$;
$vP = vP \cup t_4$;

⇒

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

Eliminate Loop Invariant Computations

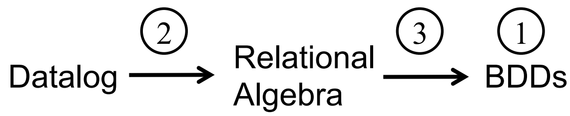
$vP'' = vP - vP'$;
$vP' = vP$;
$assign'' = assign - assign'$;
$assign' = assign$;
$t_1 = \rho_{variable \to source}(vP'')$;
$t_2 = assign \bowtie t_1$;
$t_5 = \rho_{variable \to source}(vP)$;
$t_6 = assign'' \bowtie t_5$;
$t_7 = t_2 \cup t_6$;
$t_3 = \pi_{source}(t_7)$;
$t_4 = \rho_{dest \to variable}(t_3)$;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

⇒

$vP'' = vP - vP'$;
$vP' = vP$;
$t_1 = \rho_{variable \to source}(vP'')$;
$t_2 = assign \bowtie t_1$;
$t_3 = \pi_{source}(t_2)$;
$t_4 = \rho_{dest \to variable}(t_3)$;
$vP = vP \cup t_4$;

NOTE: assign never changes

Outline



3. Datalog → BDDs

Datalog	BDDs
Relations	Boolean functions
Relation algebra: ⋈, ∪, select, project	Boolean function ops: apply, restrict, exists
Relation at a time	Function at a time
Semi-naïve evaluation	Incrementalization
Fixed-point	Iterate until stable

BDD: Relational Product (relprod)

- Relprod is a Quantified Boolean Formula
  $h = \exists x_1, x_2, .. f(x_1, x_2, ...) \wedge g(x_1, x_2, ...)$
- $h(v_1, \ldots v_n)$ is true if
  $\exists x_1, x_2, .. f(x_1, x_2, ..., v_i, ...) \wedge g(x_1, x_2, ..., v_j, ...)$
- Same as an $\wedge$ 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

$vP'' = vP - vP'$;
$vP' = vP$;
$t_1 = \rho_{variable \to source}(vP'')$;
$t_2 = assign \bowtie t_1$;
$t_3 = \pi_{source}(t_2)$;
$t_4 = \rho_{dest \to variable}(t_3)$;
$vP = vP \cup t_4$;

relprod: relational product

vP'' = diff(vP, vP');
vP' = copy(vP);
t_1 = replace(vP'',variable→source);
t_3 = relprod(t_1,assign,source);
t_4 = replace(t_3,dest→variable);
vP = or(vP, t_4);

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.

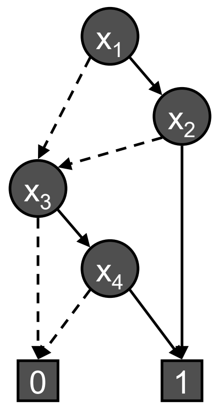
- Results explicitly generated for every clone.
- Individual results retrievable with Datalog query.

Size of BDDs

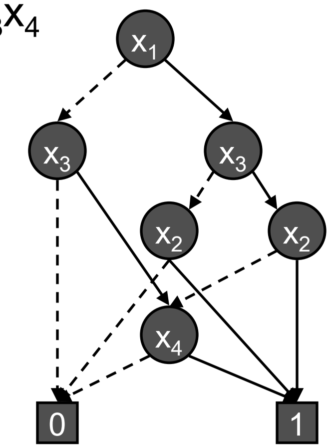
- Represent tiny and huge relations compactly
- Size depends on redundancy
  - Similar contexts have similar numberings
  - Variable ordering in BDDs

BDD Variable Order is Important!

$x_1x_2 + x_3x_4$

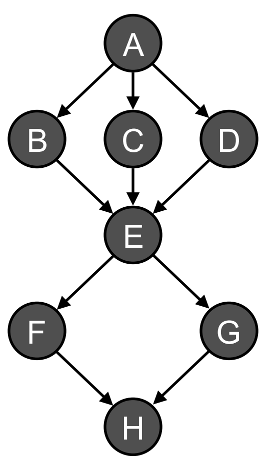


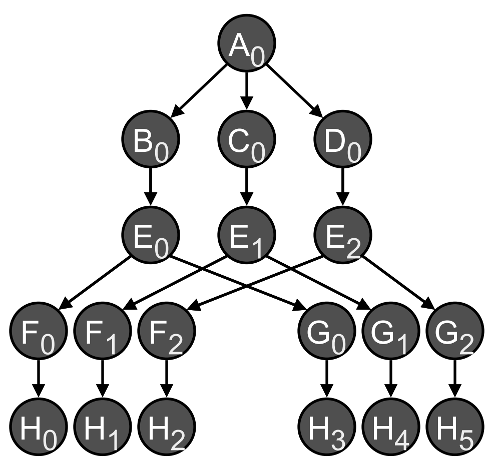
x_1,x_2,x_3,x_4



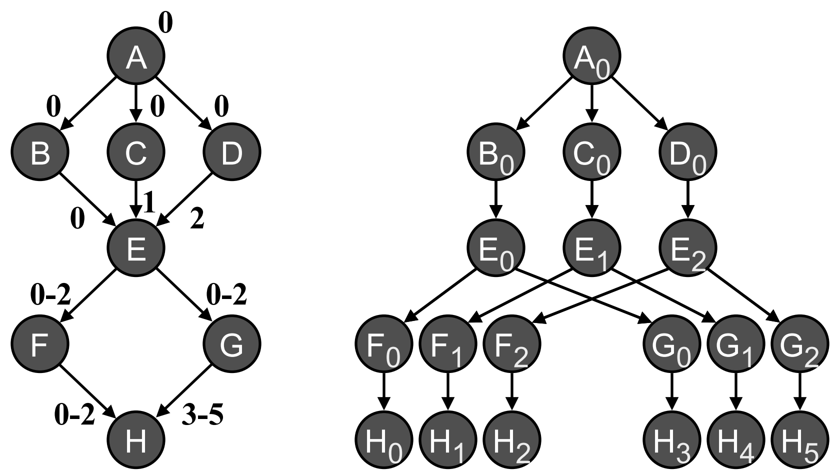
x_1,x_3,x_2,x_4

Expanded Call Graph





Numbering Clones



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. *Time: ∞*
- Improved variable ordering
  - Group similar BDD variables together.
  - Interleave equivalence relations.
  - Move common subsets to edges of variable order. *Time: 40h*
- Incrementalize outermost loop
  - Very tricky, many bugs. *Time: 36h*
- Factor away control flow, assignments
  - Reduces number of variables *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 → 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

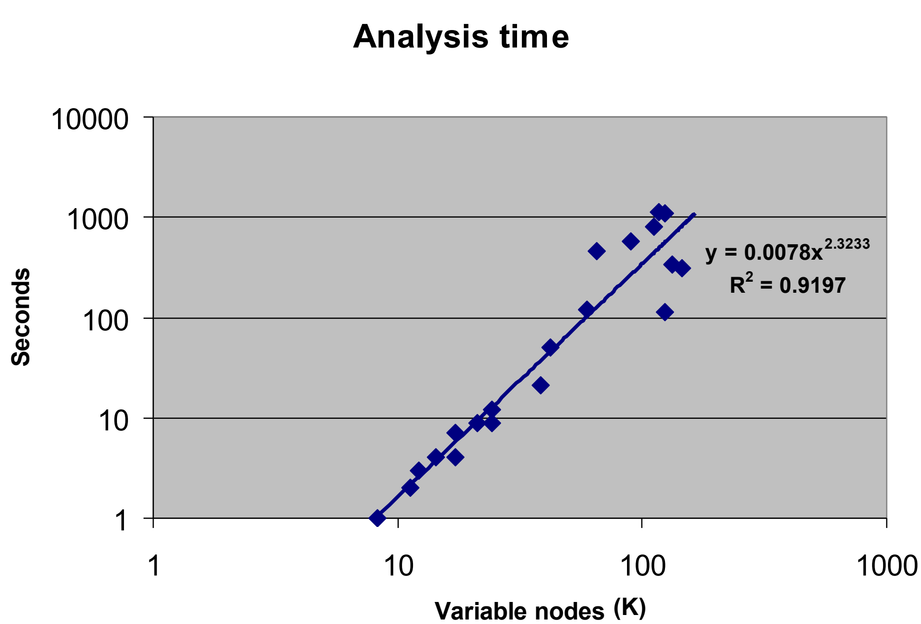
- Must be determined dynamically
- Limit trials with properties of relations
- Each trial may take a long time
- Active learning:
  select trials based on uncertainty
- Several hours
- Comparable to exhaustive for small apps

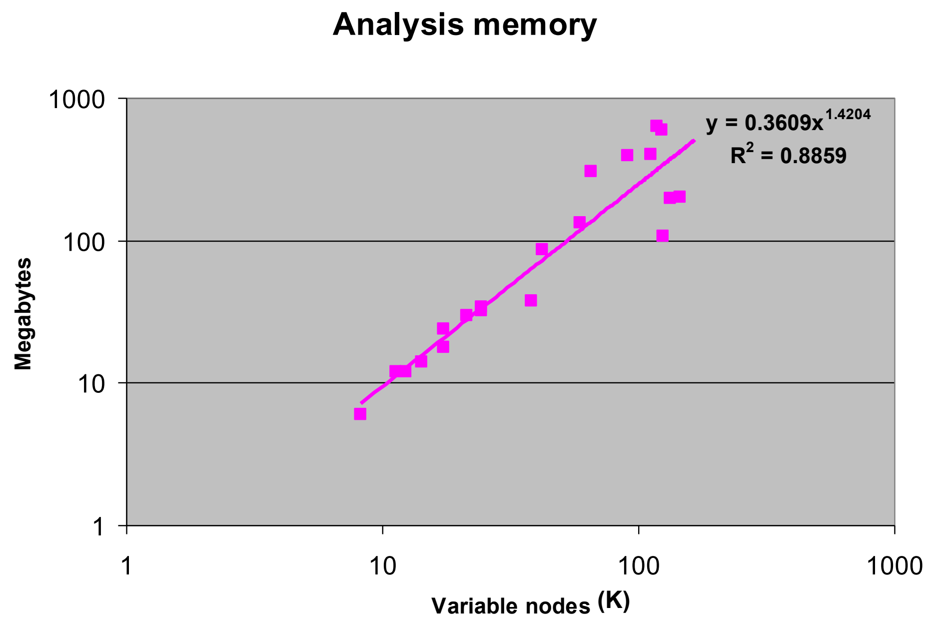
Summary: Optimizations in bddbddb

- 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 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





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

	SQL injection	HTTP splitting	Cross-site scripting	Path traveral	Total
Header	0	6	4	0	**10**
Parameter	6	5	0	2	**13**
Cookie	1	0	0	0	**1**
Non-Web	2	0	0	3	**5**
Total	**9**	**11**	**4**	**5**	**29**

Accuracy

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

Automatic Analysis Generation

Programmer:
Security analysis
in 10 lines

PQL

↓

Compiler writer:
Ptr analysis
in 10 lines

Datalog

↓ **bddbddb** (**BDD-b**ased **d**eductive **data**base**)** with Active Machine Learning

1000s of lines
1 year tuning

BDD operations

BDD: 10,000s-lines library

Software

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