# 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

**PQL**

↓

Compiler Writer:
Ptr analysis in 10 lines

**Datalog**

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

1000s of lines
1 year tuning

**BDD operations**

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

# Interprocedural Pointer Analysis

**Object creation**

pts(v, h) :- "h: T v = new T()".

**Assignment**

$pts(v_1, h_1)$ :- "$v_1 = v_2$" & $pts(v_2, h_1)$.

**Store**

$hpts(h_1, f, h_2)$ :- "$v_1.f = v_2$" & $pts(v_1, h_1)$ & $pts(v_2, h_2)$.

**Load**

$pts(v_2, h_2)$ :- "$v_2 = v_1.f$" & $pts(v_1, h_1)$ & $hpts(h_1, f, h_2)$.

**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
- 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

| | |
|---|---|
| $\cup$ | 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

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

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

$t_1 = \rho_{variable \rightarrow source}(vP);$

$t_2 = assign \bowtie t_1; \quad // (v_1, v_2, o)$

$t_3 = \pi_{source}(t_2); \quad // (v_1, o)$

$t_4 = \rho_{dest \rightarrow variable}(t_3);$

$vP = vP \cup t_4;$

# 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 – 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, 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

NOTE: assign never changes

vP'' = vP – vP';

vP' = vP;

assign'' = assign – assign';

assign' = assign;

$t_1 = \rho_{variable \rightarrow source}(vP'');$

$t_2 = assign \bowtie t_1;$

$t_5 = \rho_{variable \rightarrow source}(vP);$

$t_6 = assign'' \bowtie t_5;$

$t_7 = t_2 \cup t_6;$

$t_3 = \pi_{source}(t_7);$

$t_4 = \rho_{dest \rightarrow variable}(t_3);$

$vP = vP \cup t_4;$

⇒

vP'' = vP – vP';

vP' = vP;

$t_1 = \rho_{variable \rightarrow source}(vP'');$

$t_2 = assign \bowtie t_1;$

$t_3 = \pi_{source}(t_2);$

$t_4 = \rho_{dest \rightarrow variable}(t_3);$

$vP = vP \cup t_4;$

# Outline
# Binary Decision Diagrams (BDDs) in Pointer Analysis

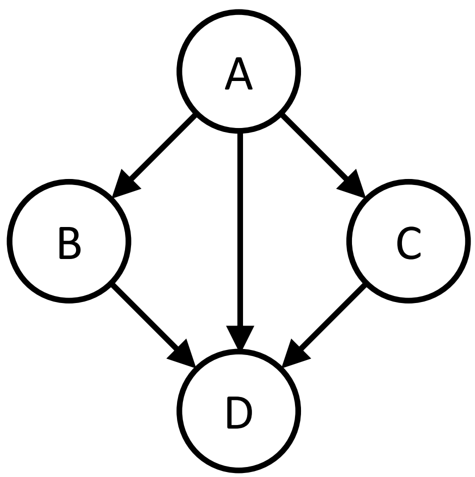
1. Datalog -> Relational Algebra
2. Relations in BDDs
3. Relational Algebra -> BDDs
4. Context-Sensitive Pointer Analysis
5. Performance of BDD Algorithms
6. Experimental Results

# 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

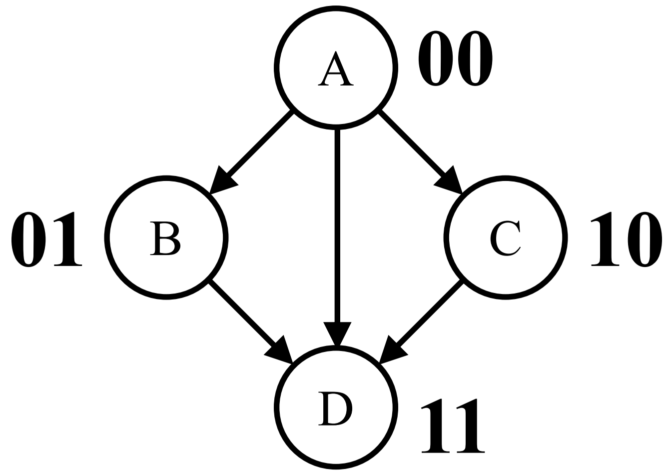



calls(A,B)
calls(A,C)
calls(A,D)
calls(B,D)
calls(C,D)

# Call Graph Relation

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | 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 |

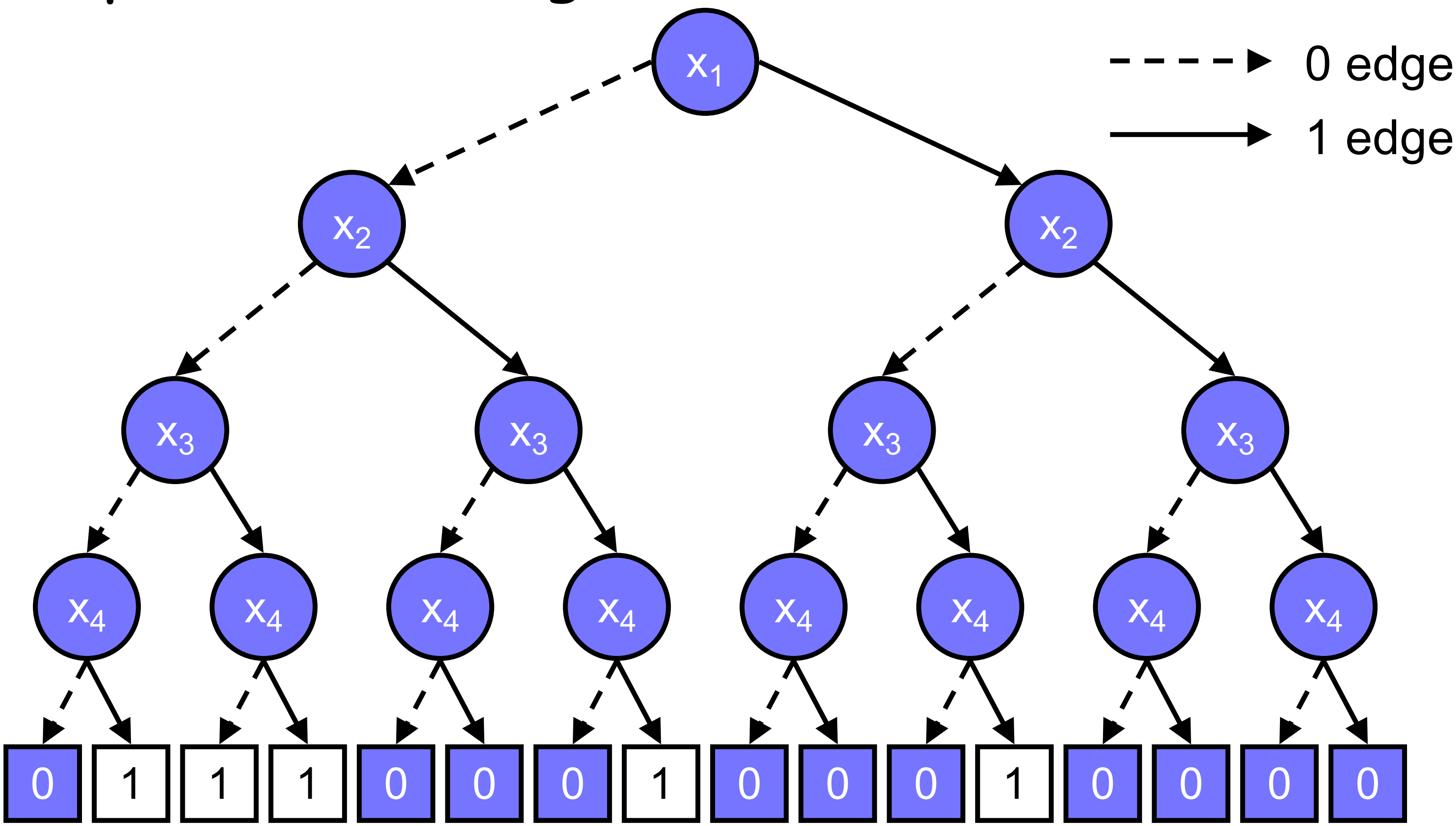
- Relation expressed as a binary function.
  - A=00, B=01, C=10, D=11




- $f(x_1,x_2,x_3,x_4)$ = calls($<x_1,x_2>$, $<x_3,x_4>$)

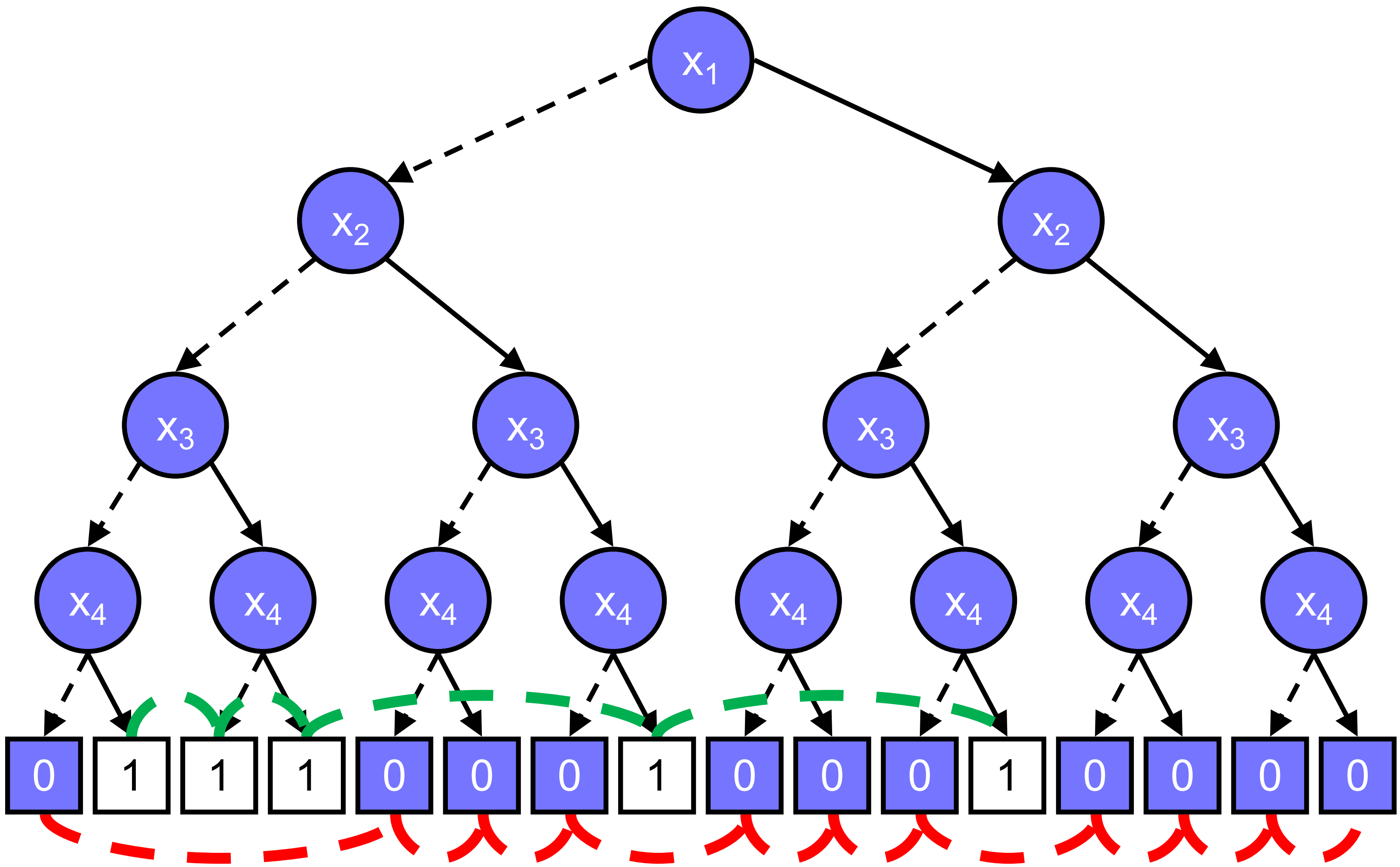
# Binary Decision Diagrams (Bryant, 1986)

- Graphical encoding of a truth table.



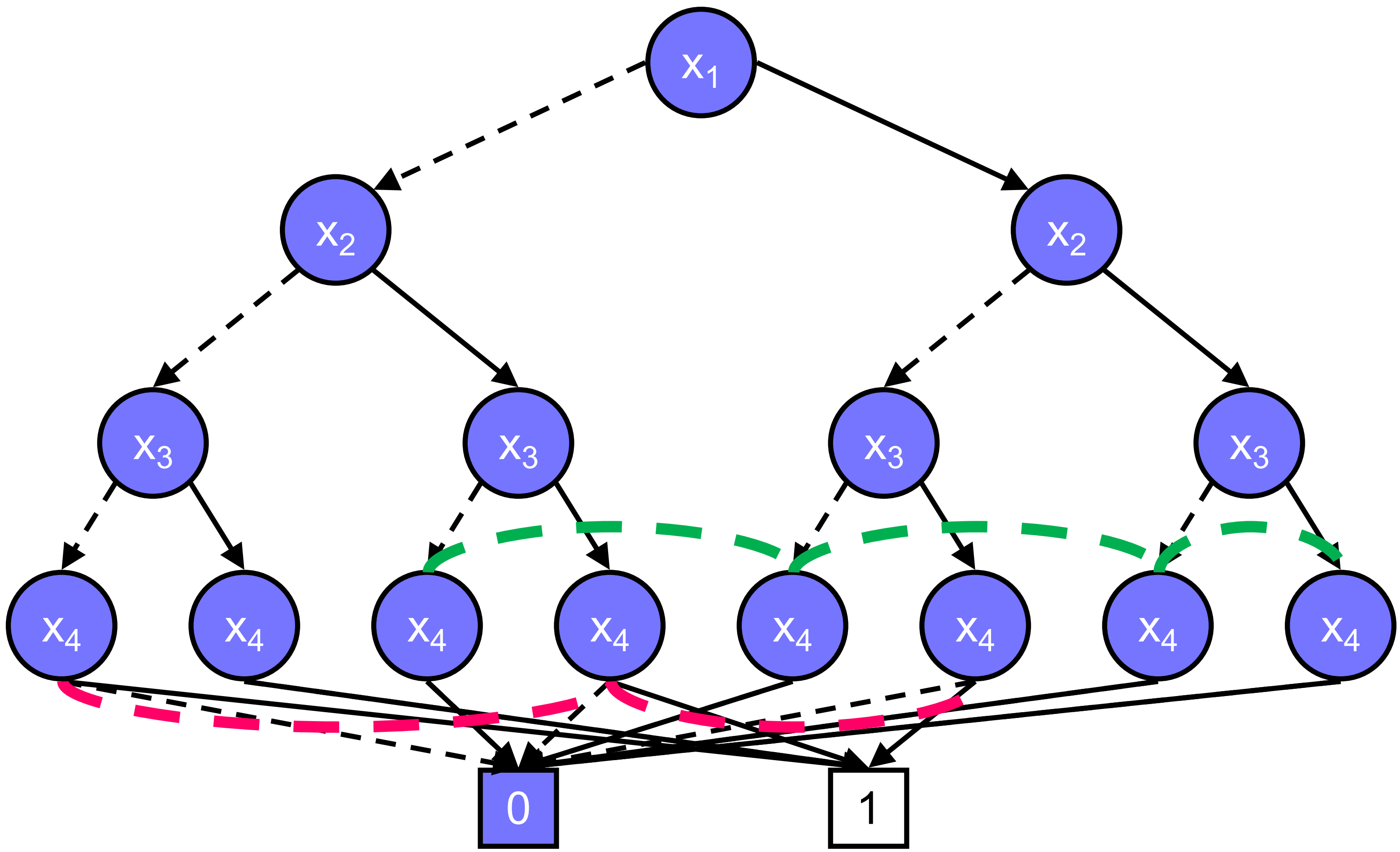
# Binary Decision Diagrams

- Collapse redundant nodes.



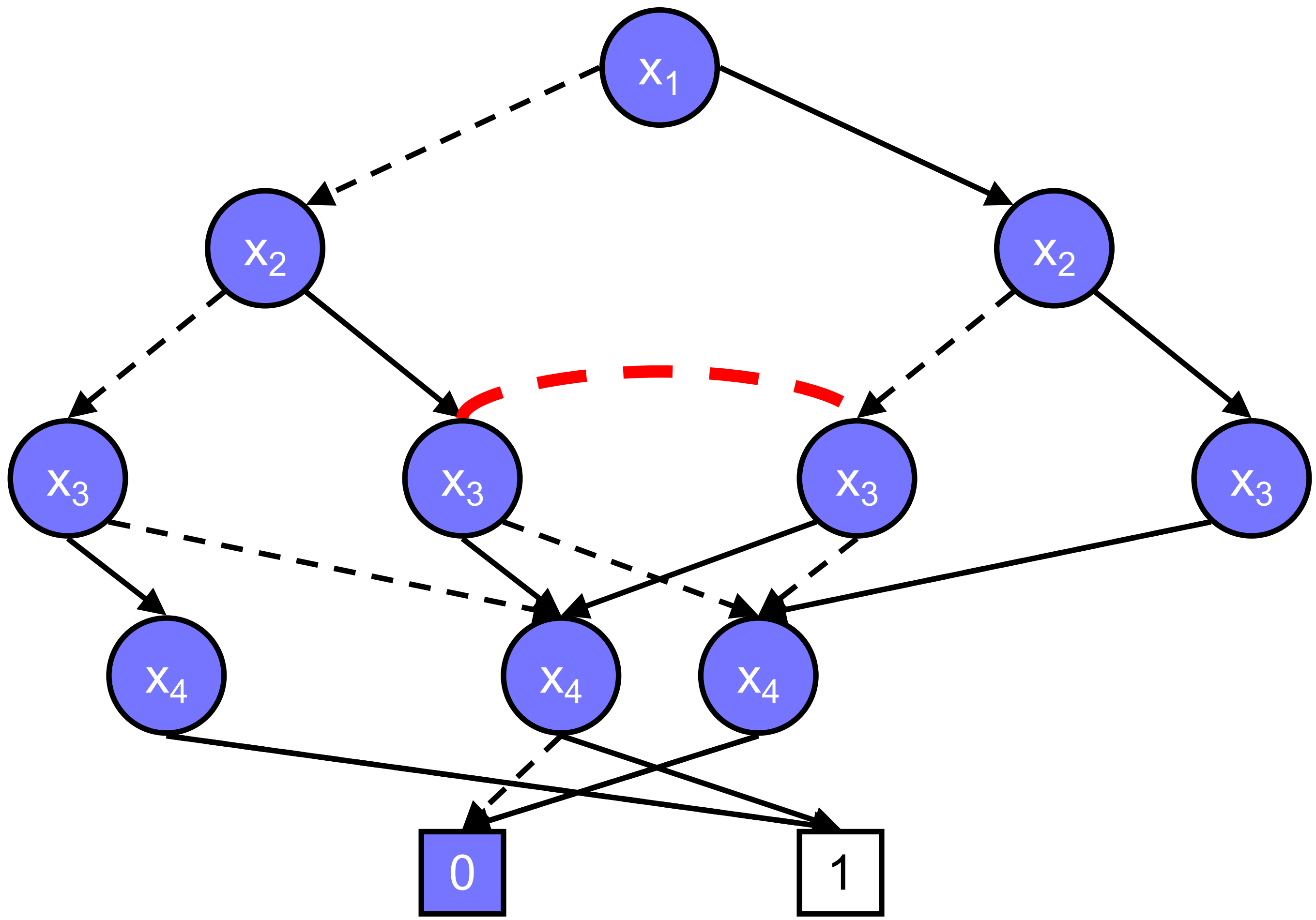
# Binary Decision Diagrams

- Collapse redundant nodes.



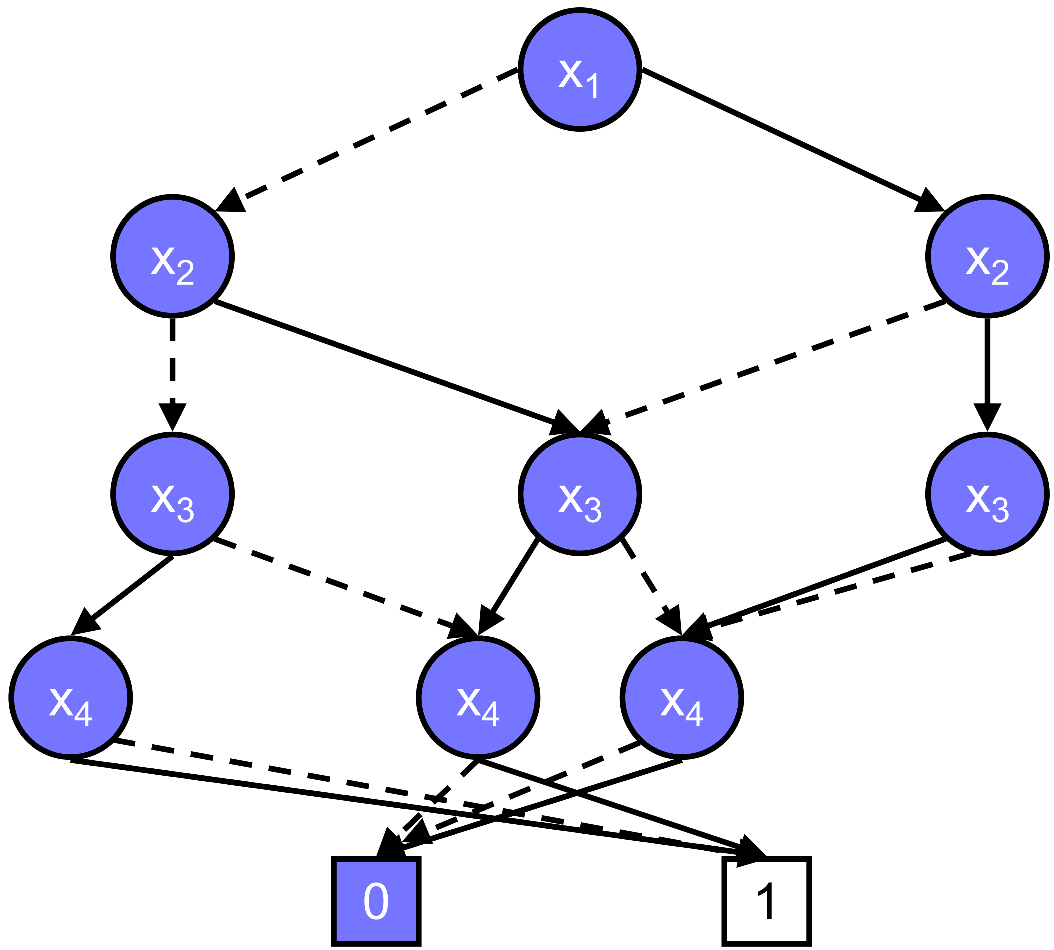
# Binary Decision Diagrams

- Collapse redundant nodes.



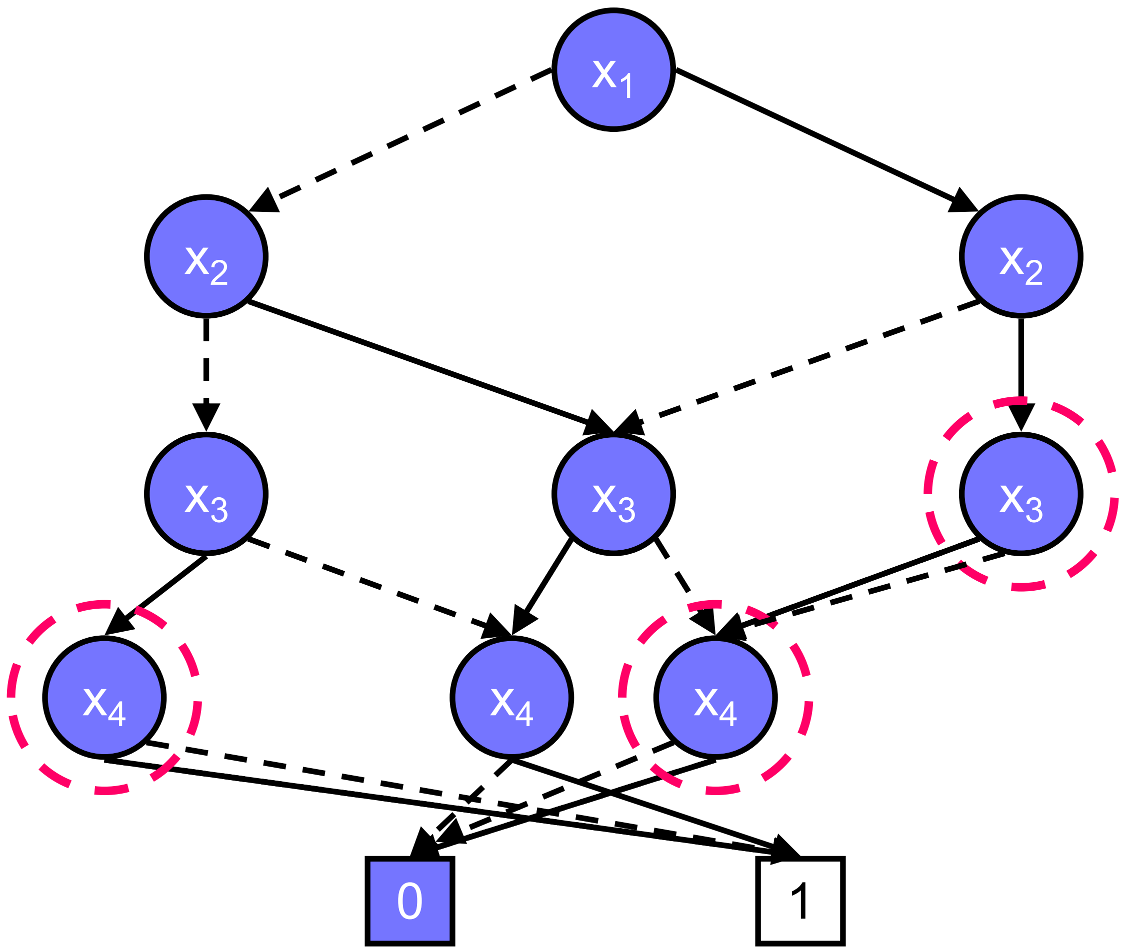
# Binary Decision Diagrams

- Collapse redundant nodes.



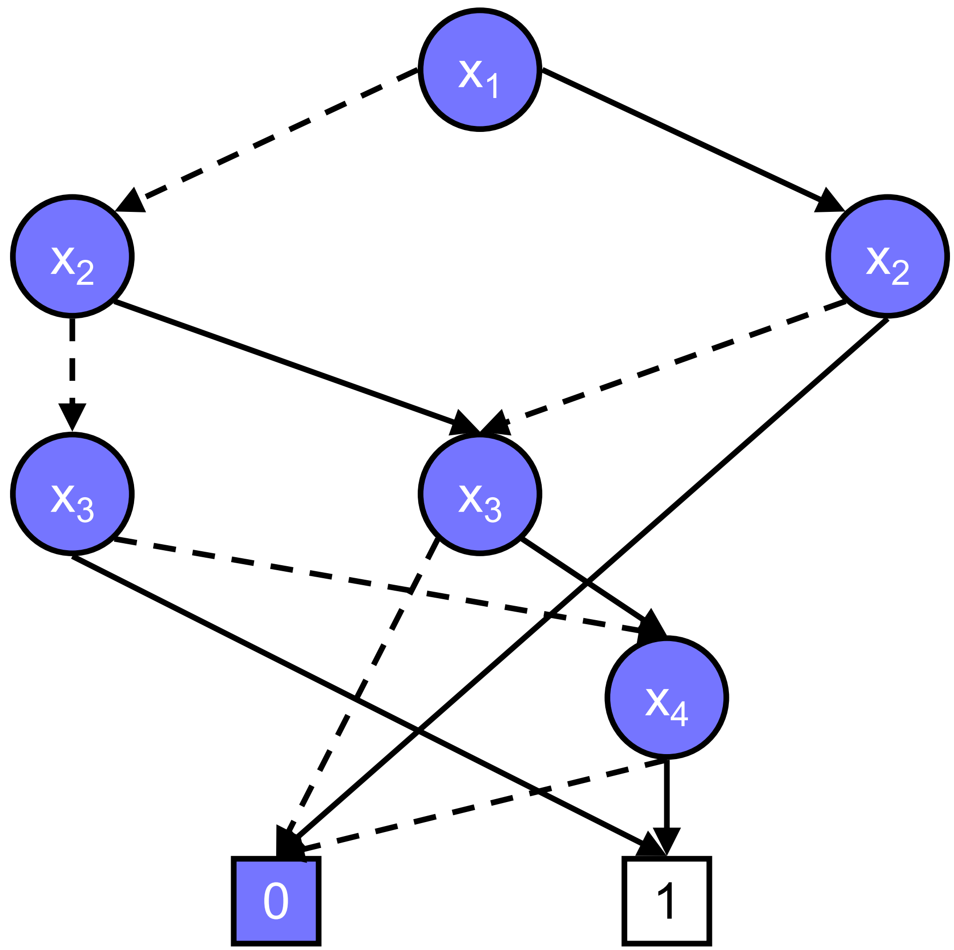
# Binary Decision Diagrams

- Eliminate unnecessary nodes.



# Binary Decision Diagrams

- Eliminate unnecessary nodes.

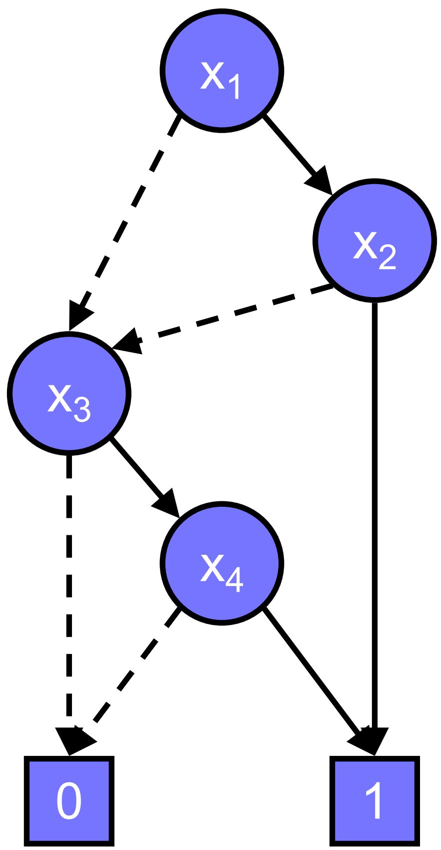


# What’s the size of

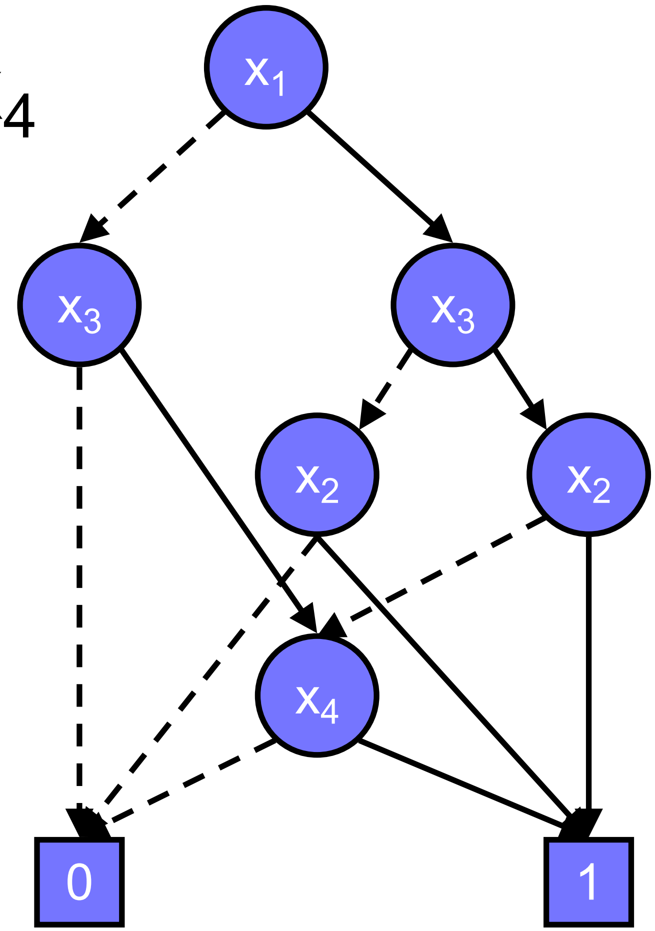
- An empty set?
- The Universal set?

# BDD Variable Order is Important to the size!

$$x_1x_2 + x_3x_4$$




$x_1,x_2,x_3,x_4$




$x_1,x_3,x_2,x_4$

# 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 → BDDs

| Datalog | BDDs |
|---|---|
| Relations | Boolean functions |
| Relation algebra: $\cup$, select, project, $\bowtie$ | Boolean function ops: apply, restrict, exists, relprod |
| Relation at a time | Function at a time |
| Semi-naïve evaluation | Incrementalization |
| Fixed-point | Iterate until stable |

# Basic 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?

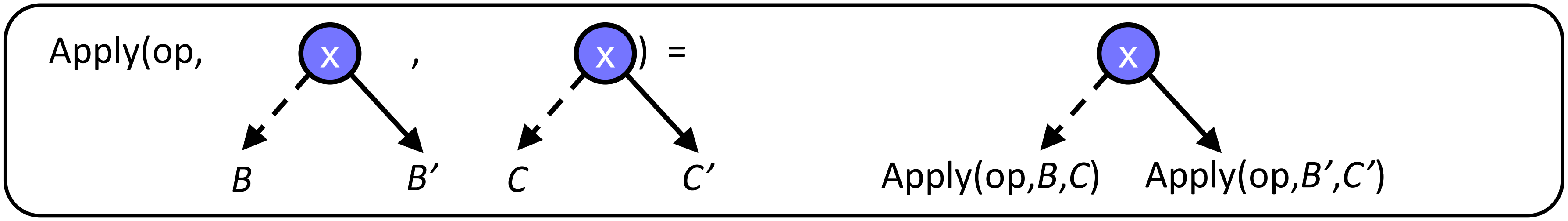
# Apply

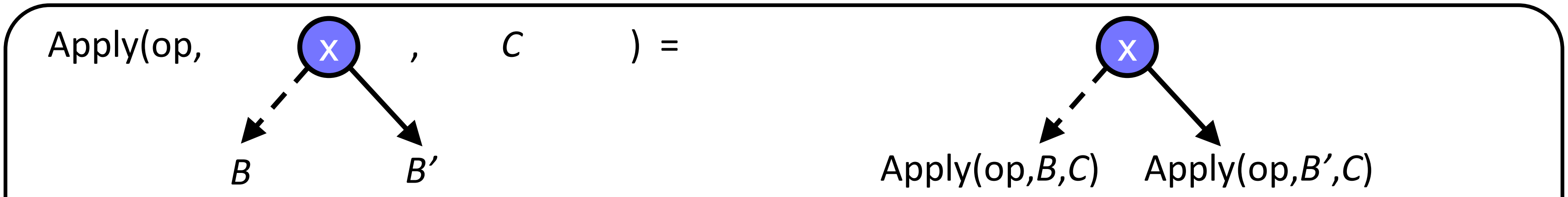
- $B$ = apply (op, $B_1$, $B_2$)
  - Combine two binary functions with a logical operator
  - $B$ is a BDD that provides the answers to all possible inputs for $B_1$ op $B_2$

# 2-input Boolean Operators: 16 Combinations

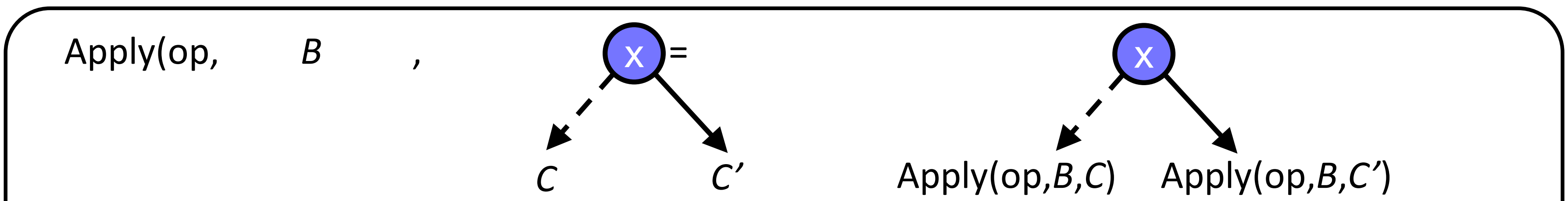
| X | 0 | 0 | 1 | 1 |
|---|---|---|---|---|
| Y | 0 | 1 | 0 | 1 |
| False | 0 | 0 | 0 | 0 |
| X and Y | 0 | 0 | 0 | 1 |
| X > Y | 0 | 0 | 1 | 0 |
| X | 0 | 0 | 1 | 1 |
| X < Y | 0 | 1 | 0 | 0 |
| Y | 0 | 1 | 0 | 1 |
| X XOR Y | 0 | 1 | 1 | 0 |
| X OR Y | 0 | 1 | 1 | 1 |
| X NOR Y | 1 | 0 | 0 | 0 |
| 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




Where *C* is (1) a terminal node or (2) a non-terminal with var (root(*C*)) > *x*

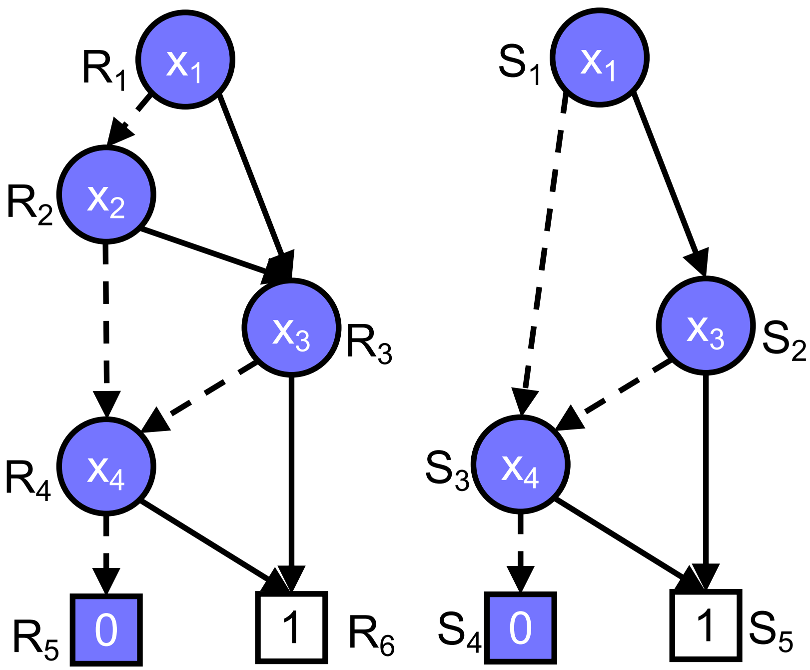



Where *B* is (1) a terminal node or (2) a non-terminal with var (root(*B*)) > *x*

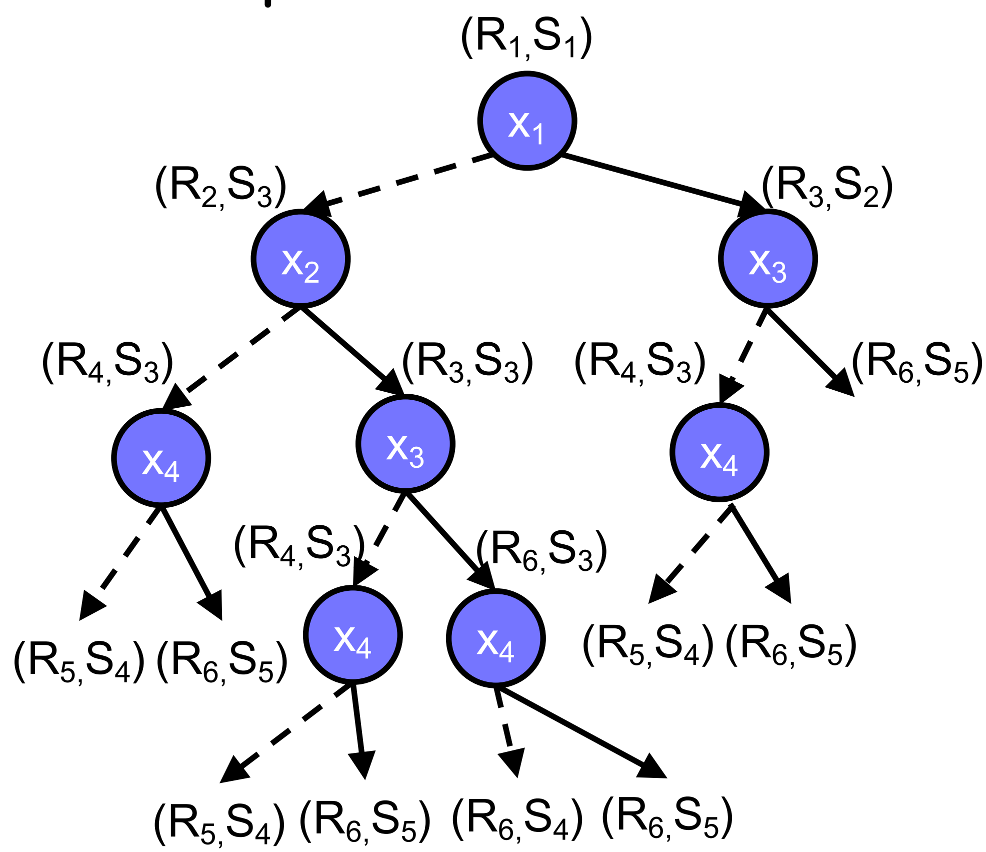
Apply(op, *u* , *v* ) = *w*, where *w* = *u* op *v*

# Example: Apply (op, R, S)

- Combine the BDDs for generic op




E1: $(x_1 \wedge x_3) \vee x_4 \vee (x_2 \wedge x_3)$ E2: $(x_1 \wedge x_3) \vee x_4$

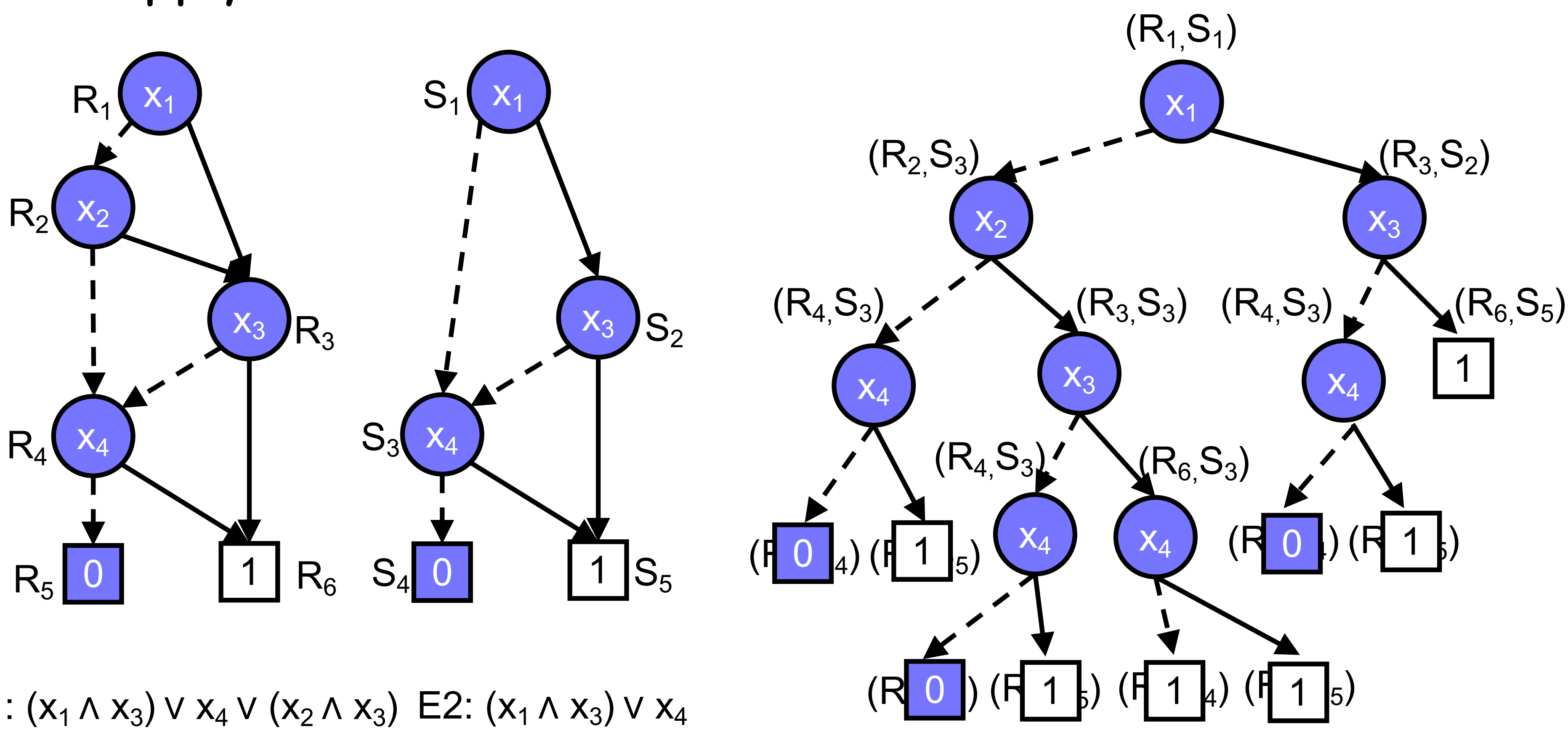



E1 op E2

Question: what is E1 v E2?

# Example: Apply (OR, R, S)

- Apply OR to the constant nodes

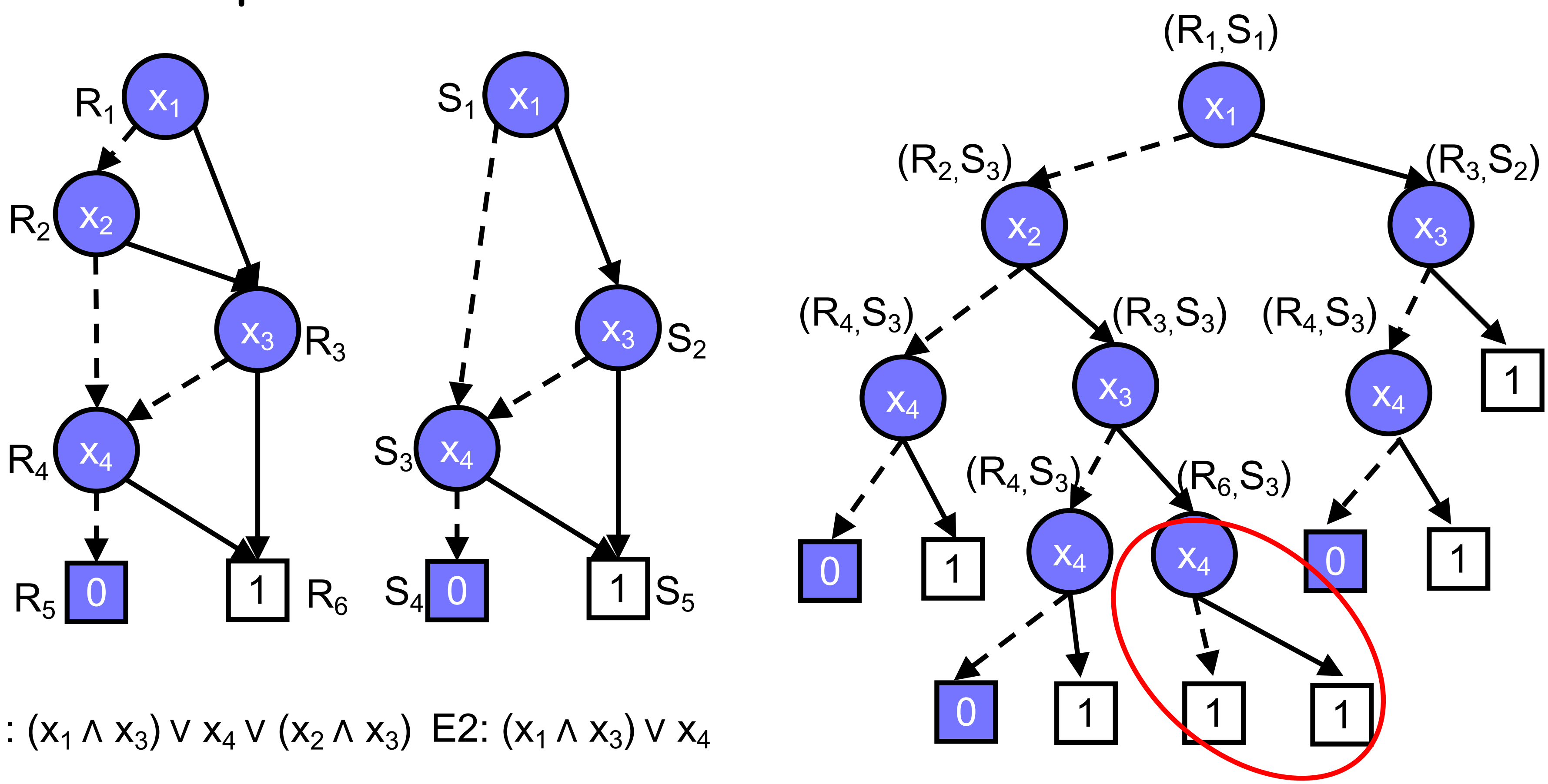



E1: $(x_1 \wedge x_3) \vee x_4 \vee (x_2 \wedge x_3)$ E2: $(x_1 \wedge x_3) \vee x_4$

E1 v E2

# Example: Apply (OR, R, S)

- Collapse redundant nodes

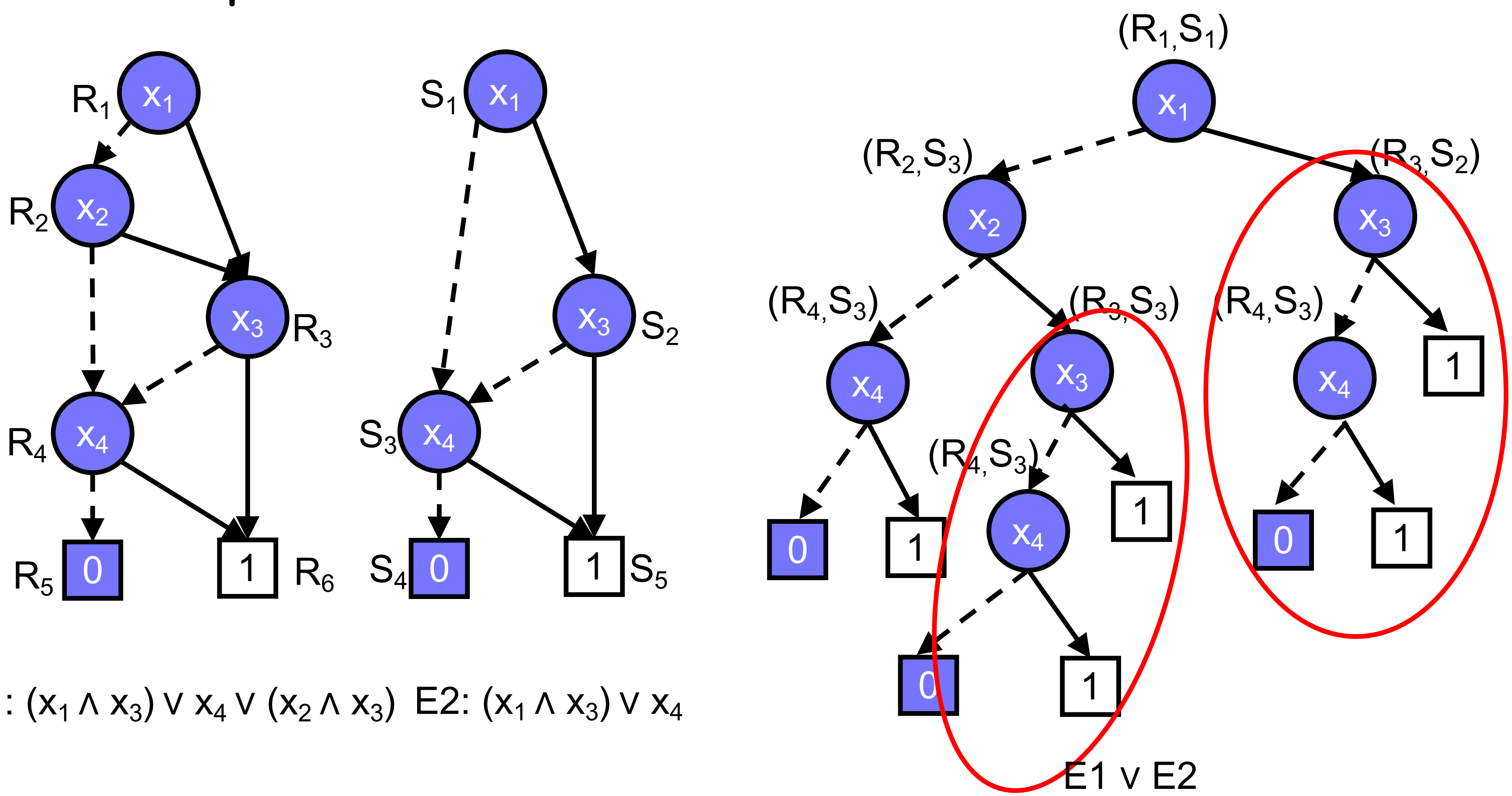



E1: $(x_1 \wedge x_3) \vee x_4 \vee (x_2 \wedge x_3)$ E2: $(x_1 \wedge x_3) \vee x_4$

E1 ∨ E2

# Example: Apply (OR, R, S)

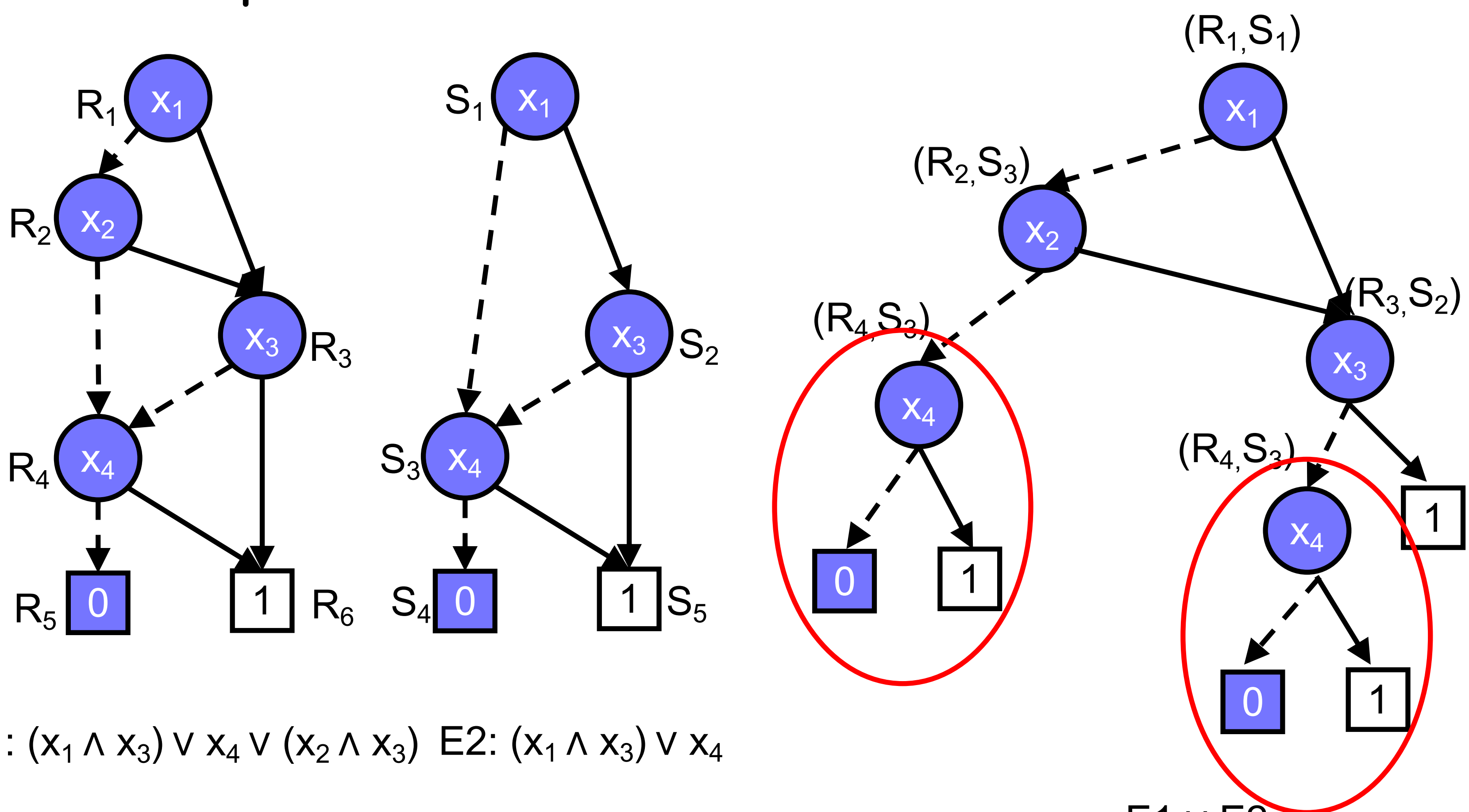
- Collapse redundant nodes




E1: $(x_1 \wedge x_3) \vee x_4 \vee (x_2 \wedge x_3)$ E2: $(x_1 \wedge x_3) \vee x_4$

# Example: Apply (OR, R, S)

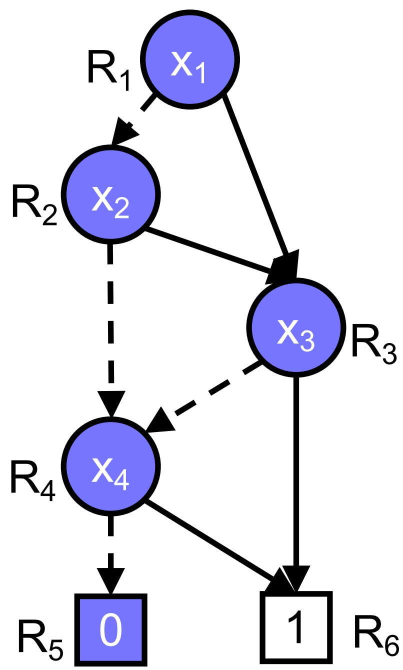
- Collapse redundant nodes

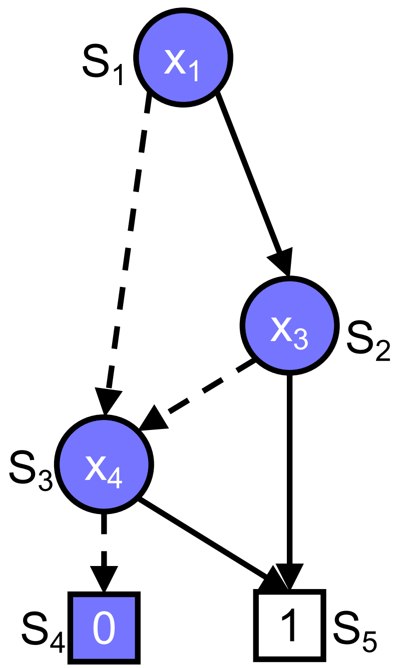


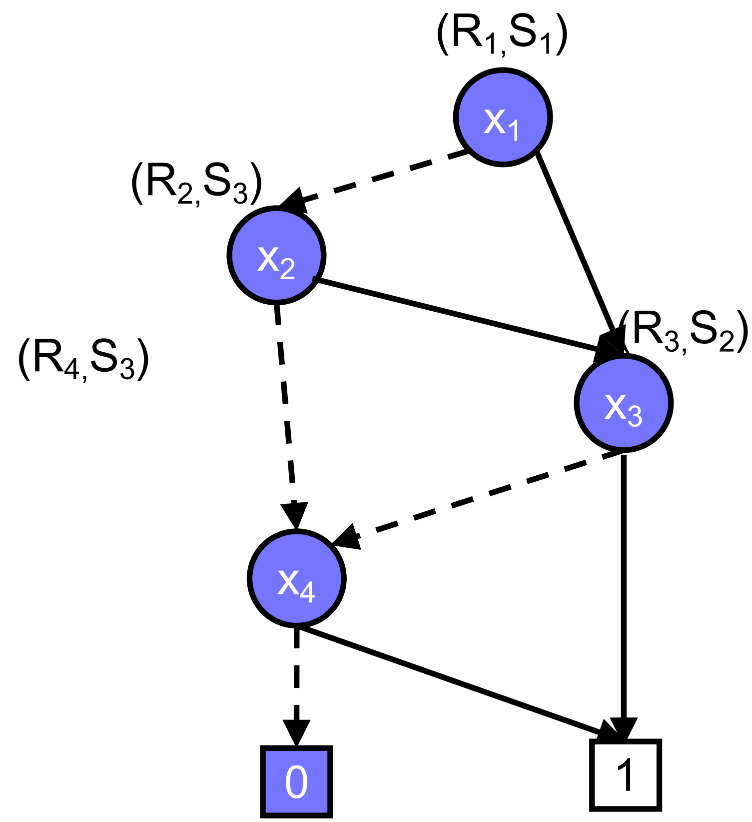

E1: $(x_1 \wedge x_3) \vee x_4 \vee (x_2 \wedge x_3)$ E2: $(x_1 \wedge x_3) \vee x_4$

E1 ∨ E2

# Example: Apply (OR, R, S)



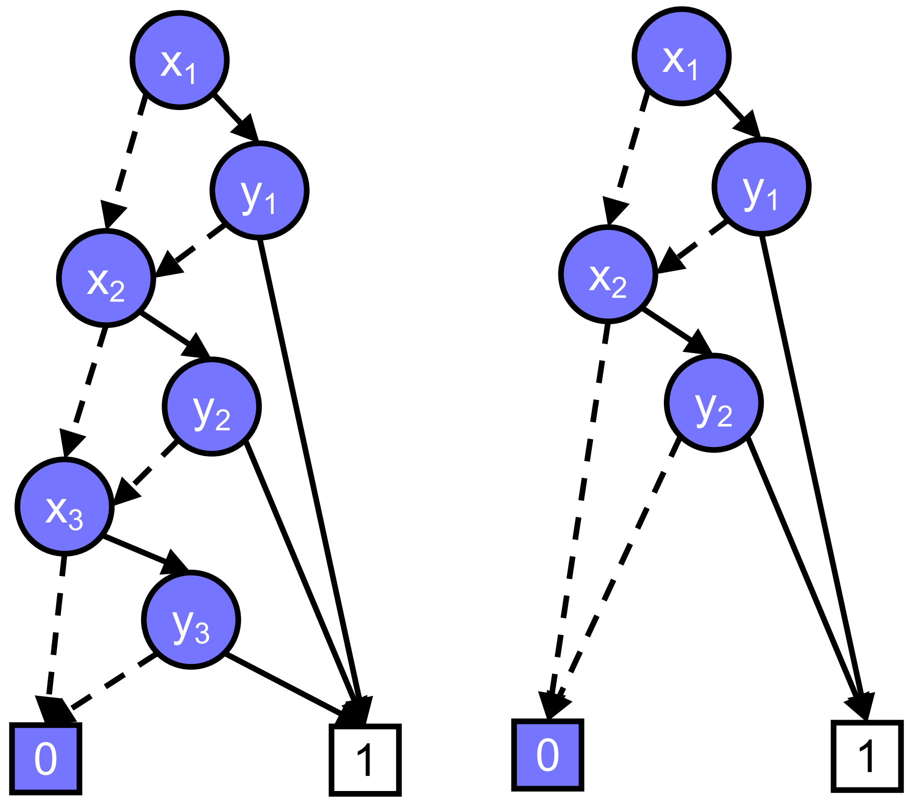

E1: $(x_1 \wedge x_3) \vee x_4 \vee (x_2 \wedge x_3)$ E2: $(x_1 \wedge x_3) \vee x_4$

E1 v E2

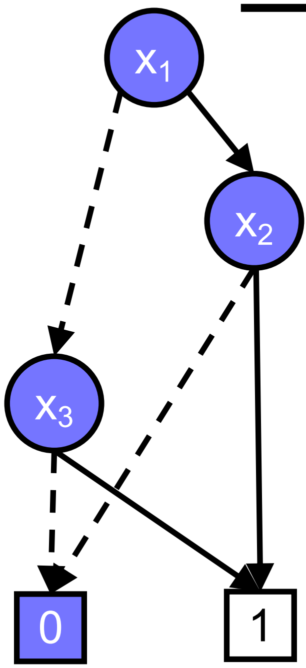
# Algorithm: Restrict

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

restrict(0, $x_3$, B)



# Algorithm: Exists




E: $(x_1 \wedge x_2) \vee (\overline{x_1} \wedge x_3)$

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

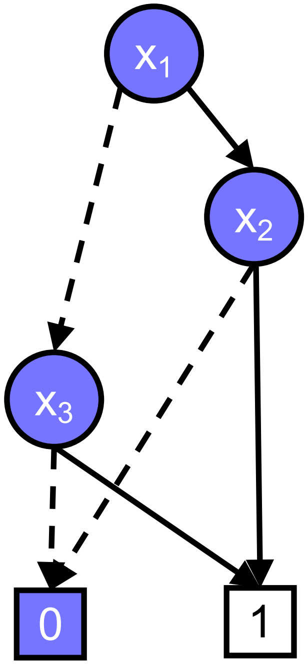
When does there exist an $x_1$ such that E is true?

Useful inference rule:

Resolve $$\frac{p \vee A \qquad \neg p \vee B}{A \vee B}$$

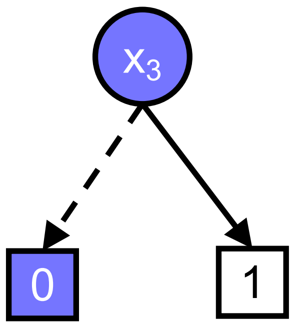
- $B_1$ = exists(x,B)
  = apply (OR, restrict (0,x,B), restrict (1,x,B))
- $B_1$ = 0 if there does not exist an x
  = binary function (without variable x)
  that defines when there exists an x
  such that B is true.

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



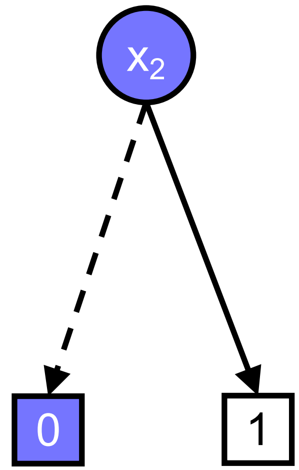

D

$(x_1 \wedge x_2) \vee (\overline{x_1} \wedge x_3)$




restrict(0, $x_1$, D)

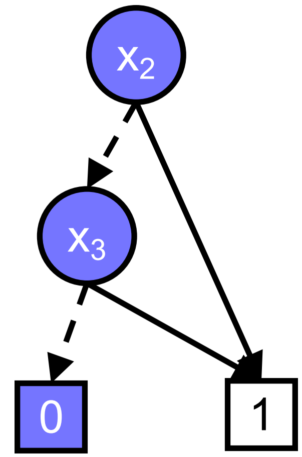
$x_3$




restrict(1, $x_1$, D)

$x_2$

When?




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

$x_2 \vee x_3$

# BDD: Relational Product (relprod)

- Relprod is a Quantified Boolean Formula
  (Corresponding to join + project in relational algebra)
- h = Relprod(f, g, $[x_1, x_2, \ldots]$)
  - $h(v_1, \ldots v_n)$ is true if
    $\exists x_1, x_2, \ldots f(x_1, x_2, \ldots, v_i, \ldots) \wedge g(x_1, x_2, \ldots, v_j, \ldots)$
- Same as an $\wedge$ operation
  followed by projecting away common attributes $x_1, x_2, \ldots$
- Important because it is common and much faster to
  combine the $\wedge$ and projection operations in BDDs

# Relational algebra -> BDD operations

$vP'' = vP - vP'$;

$vP' = vP$;

$t_1 = \rho_{variable \to source}(vP'')$;

$t_2 = assign \bowtie t_1$;

$t_3 = \pi_{source}(t_2)$;

$t_4 = \rho_{dest \to variable}(t_3)$;

$vP = vP \cup t_4$;

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, $t_4$);

# 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

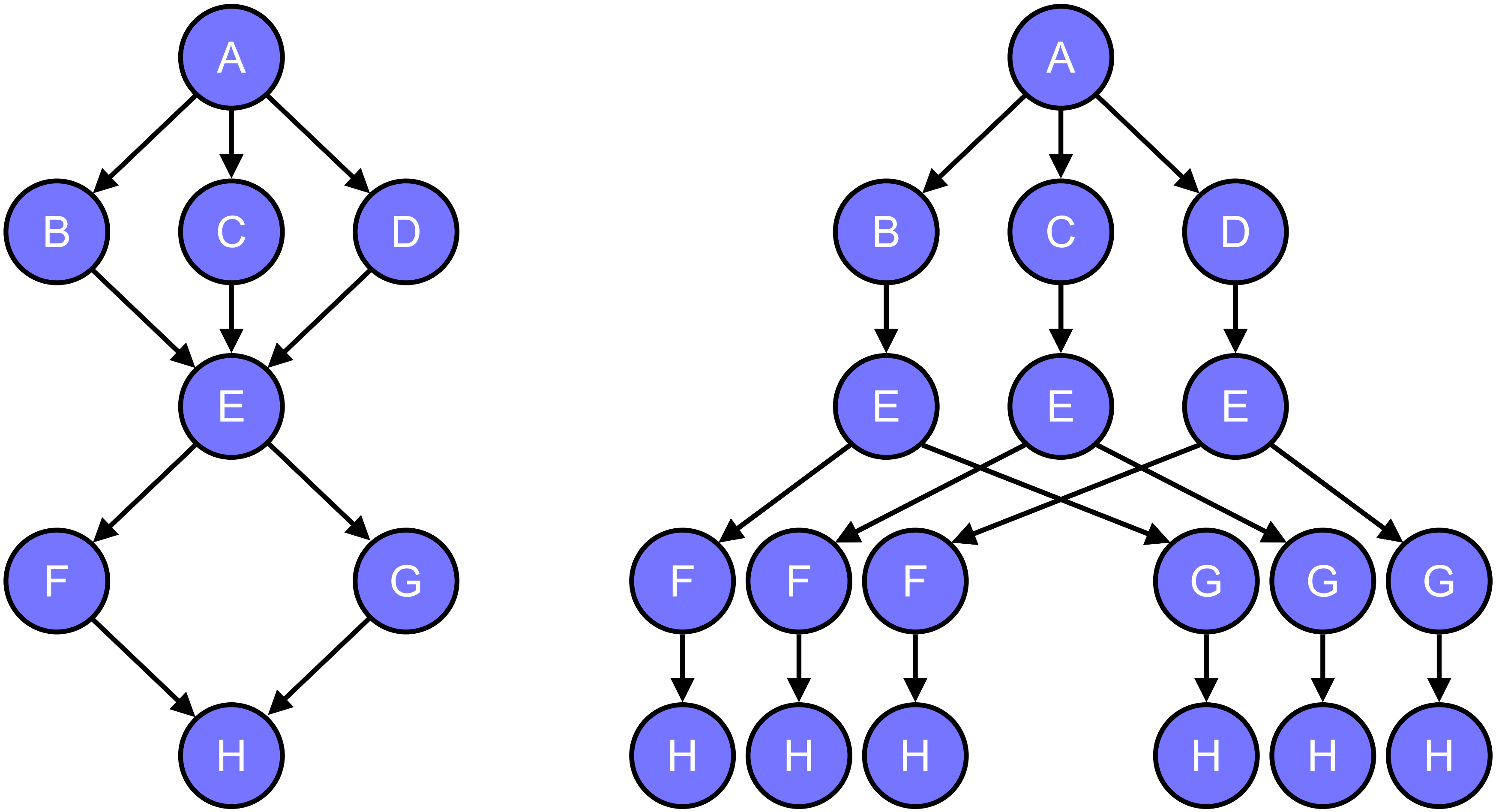
1. First, do context-insensitive pointer analysis to get call graph.
2. Number clones.
3. Do context-insensitive algorithm on the cloned graph.

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

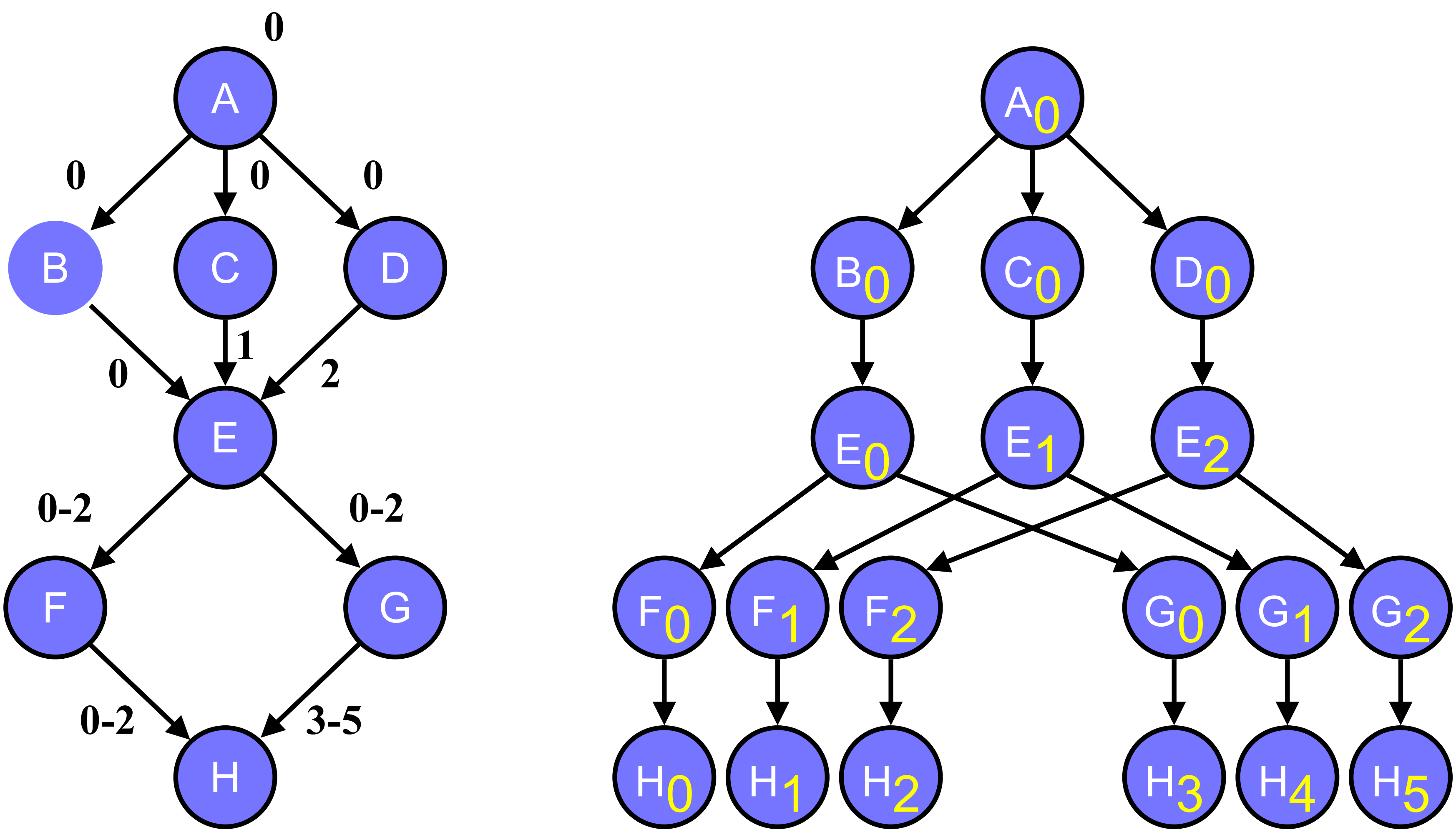
# Size of BDDs

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

# Expanded Call Graph



# Numbering Clones



# 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. *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

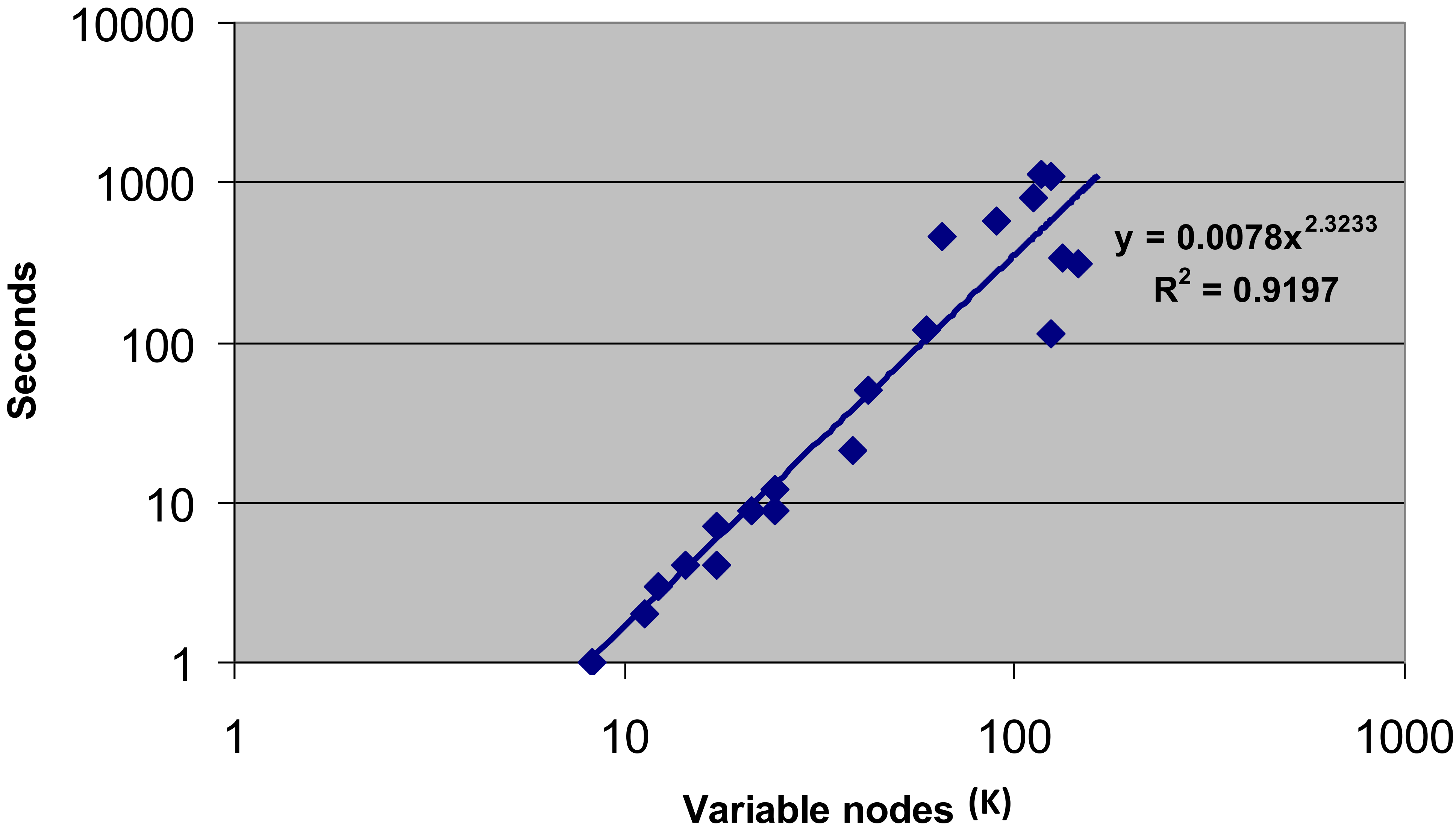
# Outline
# Binary Decision Diagrams (BDDs) in Pointer Analysis

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

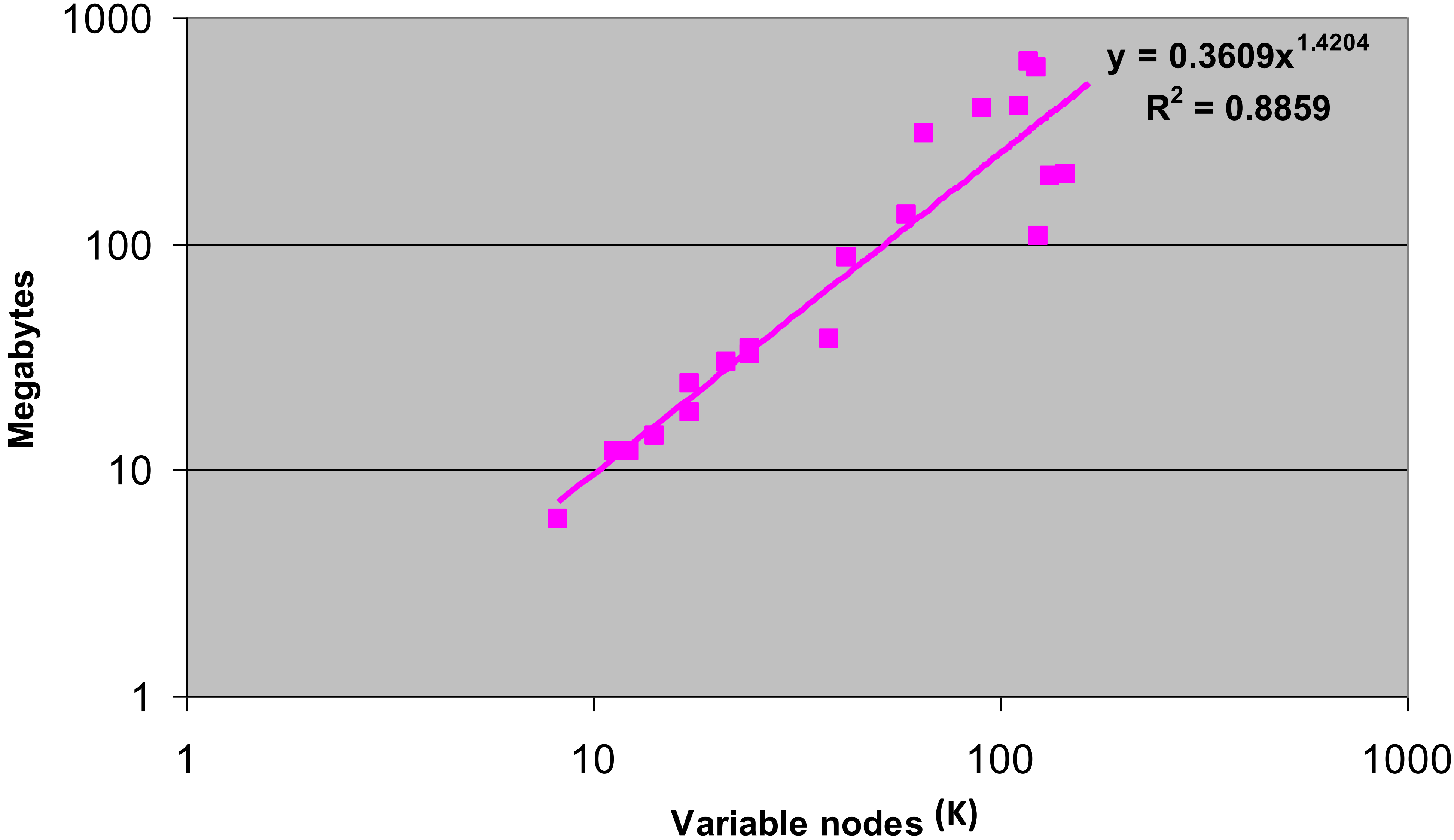
# 6. Experimental Results

- 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



# Analysis memory



# 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 traversal | 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 Conservative Analysis Generation

Programmer:
Security analysis in 10 lines

PQL

Compiler Writer:
Ptr analysis in 10 lines

Datalog

**bddbddb**
(**BDD**-**b**ased
**d**eductive **d**ata**b**ase)
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