Pointer Analysis – Part II

CS 6340

Unification vs. Inclusion

- Earlier scalable pointer analysis was context-insensitive unification-based [Steensgaard '96]
  - Pointers are either un-aliased or point to the same set of objects
  - Near-linear, but very imprecise
- Inclusion-based pointer analysis
  - Can point to overlapping sets of objects
  - Closure calculation is $O(n^3)$
  - Various optimizations [Fahndrich, Su, Heintze,…]
  - BDD formulation, simple, scalable [Berndl, Zhu]

Context Sensitivity

- Context sensitivity is important for precision
  - Unrealizable paths

```
a = id(b);                    c = id(d);

Object id(Object x) {
    return x;
}
```


Context Sensitivity

- Context sensitivity is important for precision
  - Unrealizable paths
  - Conceptually give each caller its own copy

```
a = id(b);                    c = id(d);

Object id(Object x) {
    return x;
}
```

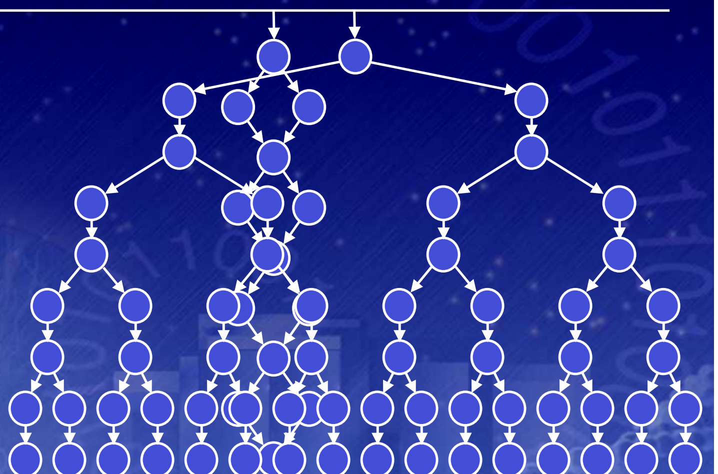
Summary-Based Analysis

- Popular method for context sensitivity
- Two kinds:
  - Bottom-up
  - Top-down
- Problems:
  - Difficult to summarize pointer analysis
  - Summary-based analysis using BDD: not shown to scale [Zhu'02]

Cloning-Based Analysis

- Simple brute force technique
  - Clone every path through the call graph
  - Run context-insensitive algorithm on the expanded call graph
- The catch: exponential blowup

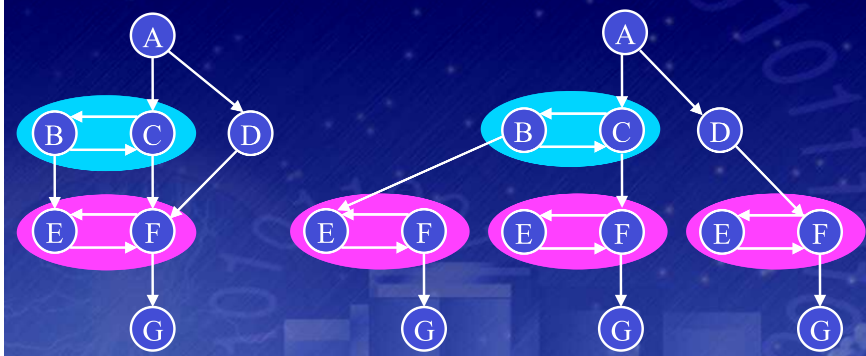
Cloning is exponential!



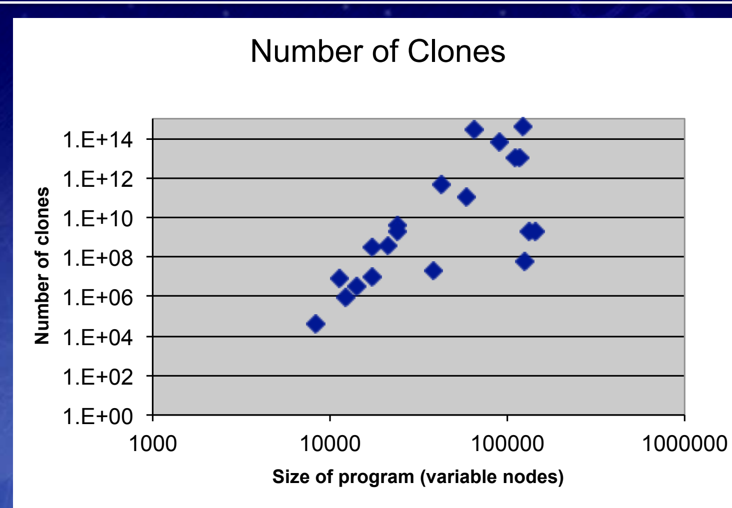
Recursion

- Actually, cloning is unbounded in the presence of recursive cycles
- Solution: treat all methods in a strongly-connected component as a single node

Recursion



Top 20 Sourceforge Java Apps



Cloning is infeasible (?)

- Typical large program has ~10^{14} paths
- If you need 1 byte to represent a clone, would require 256 terabytes of storage
  - Registered ECC 1GB DIMMs: $98.6 million
    - Power: 96.4 kilowatts = Power for 128 homes
  - 300 GB hard disks: 939 x $250 = $234,750
    - Time to read (sequential): 70.8 days
- Seems unreasonable!

BDD comes to the rescue

- There are many similarities across contexts
  - Many copies of nearly-identical results
- BDDs can represent large sets of redundant data efficiently
  - Need a BDD encoding that exploits the similarities

Contribution (1)

- Can represent context-sensitive call graph efficiently with BDDs and a clever context numbering scheme
  - Inclusion-based pointer analysis
    - 10^{14} contexts, 19 minutes
  - Generates all answers

Contribution (2)

BDD hacking is complicated → bddbddb (BDD-based deductive database)

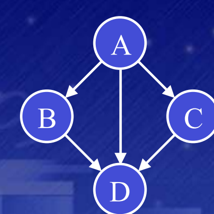
- Pointer analysis in 6 lines of Datalog
- Automatic translation into efficient BDD implementation
- 10x performance over hand-tuned solver (2164 lines of Java)

Contribution (3)

- bddbddb: General Datalog solver
  - Supports simple declarative queries
  - Easy use of context-sensitive pointer results
- Simple context-sensitive analyses:
  - Escape analysis
  - Type refinement
  - Side effect analysis
  - Many more presented in the paper

Call Graph Relation

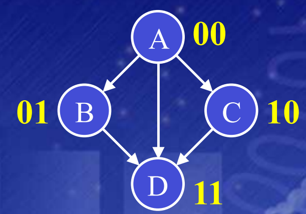
- Call graph expressed as a relation
  - Five edges:
    - Calls(A,B)
    - Calls(A,C)
    - Calls(A,D)
    - Calls(B,D)
    - Calls(C,D)



Call Graph Relation

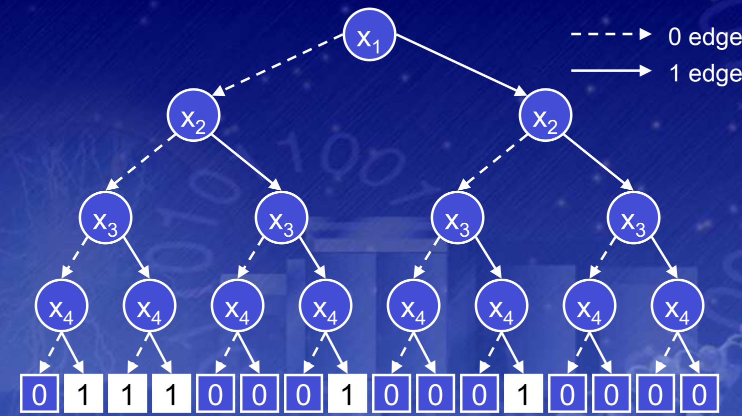
x_1	x_2	x_3	x_4	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

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



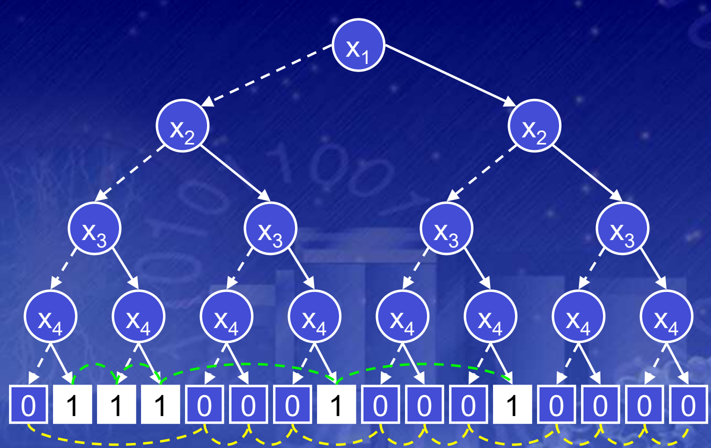
Binary Decision Diagrams

- Graphical encoding of a truth table



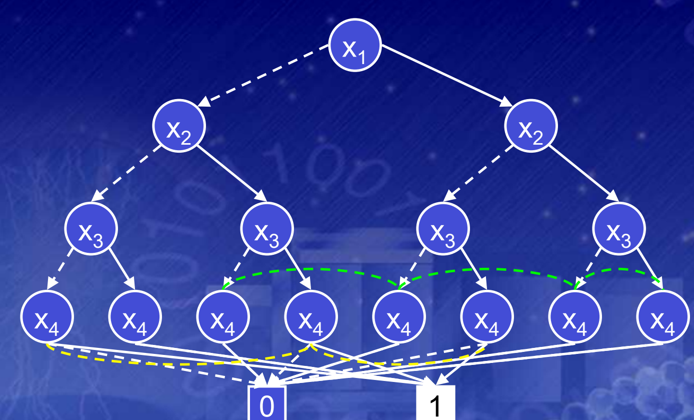
Binary Decision Diagrams

- Collapse redundant nodes



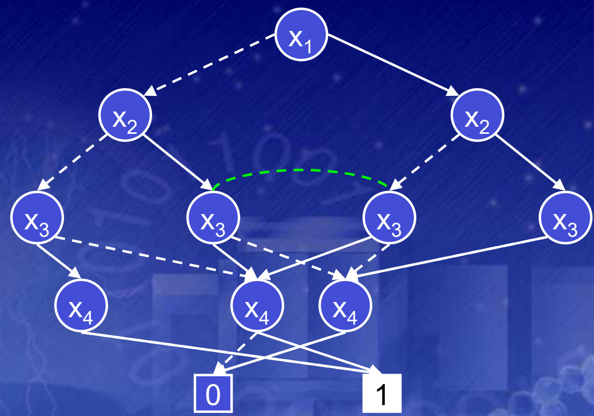
Binary Decision Diagrams

- Collapse redundant nodes



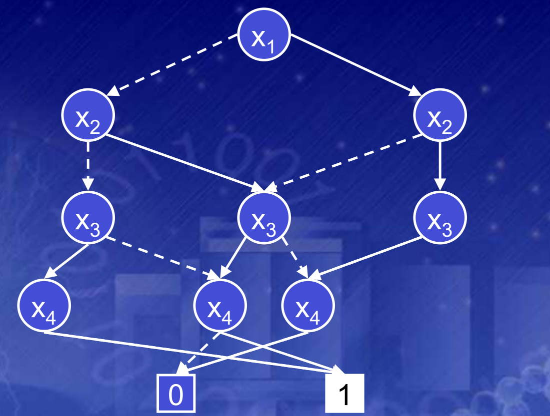
Binary Decision Diagrams

- Collapse redundant nodes



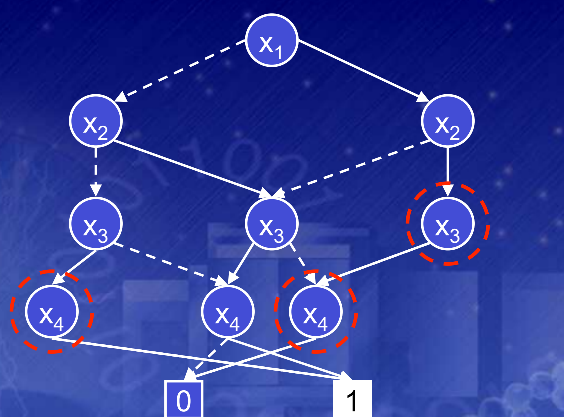
Binary Decision Diagrams

- Collapse redundant nodes



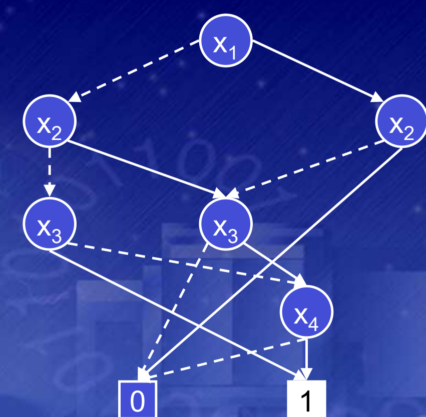
Binary Decision Diagrams

- Eliminate unnecessary nodes



Binary Decision Diagrams

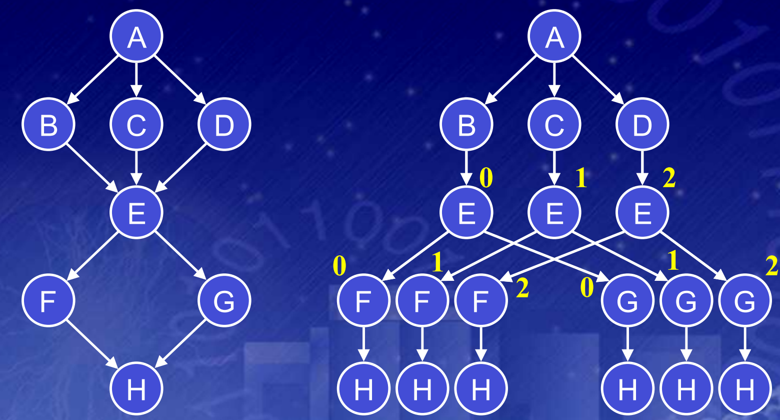
- Eliminate unnecessary nodes



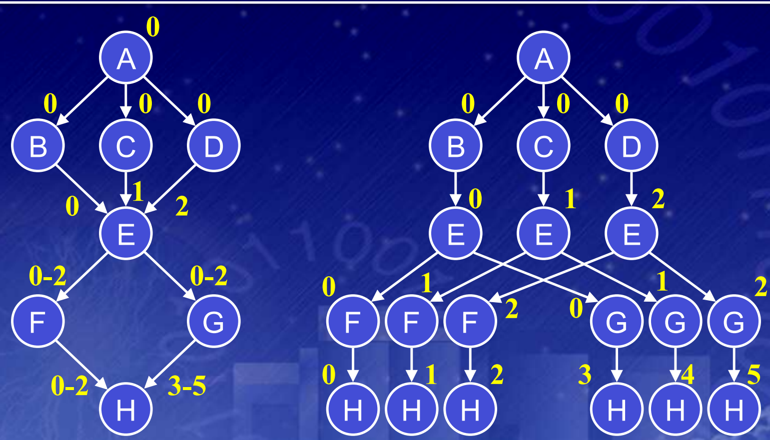
Binary Decision Diagrams

- Size is correlated to amount of redundancy, NOT size of relation
  - As the set gets larger, the number of don't-care bits *increases*, leading to fewer necessary nodes

Expanded Call Graph



Numbering Clones

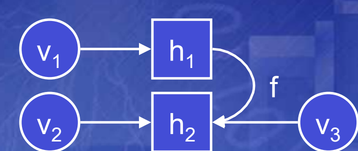


Pointer Analysis Example

h_1: v_1 = new Object();
h_2: v_2 = new Object();
v_1.f = v_2;
v_3 = v_1.f;

<u>Input Relations</u>

vPointsTo(v_1, h_1)
vPointsTo(v_2, h_2)
Store(v_1, f, v_2)
Load(v_1, f, v_3)



<u>Output Relations</u>

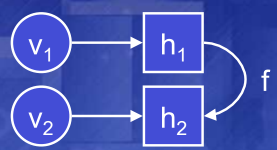
hPointsTo(h_1, f, h_2)
vPointsTo(v_3, h_2)

Inference Rule in Datalog

Heap Writes:

hPointsTo(h_1, f, h_2) :- Store(v_1, f, v_2),
vPointsTo(v_1, h_1),
vPointsTo(v_2, h_2).

v_1.f = v_2;



Context-sensitive pointer analysis

- Compute call graph with context-insensitive pointer analysis
  - Datalog rules for:
    - assignments, loads, stores
    - discover call targets, bind parameters
    - type filtering
  - Apply rules until fix-point reached
- Compute expanded call graph relation
- Apply context-insensitive algorithm to the expanded call graph

Datalog

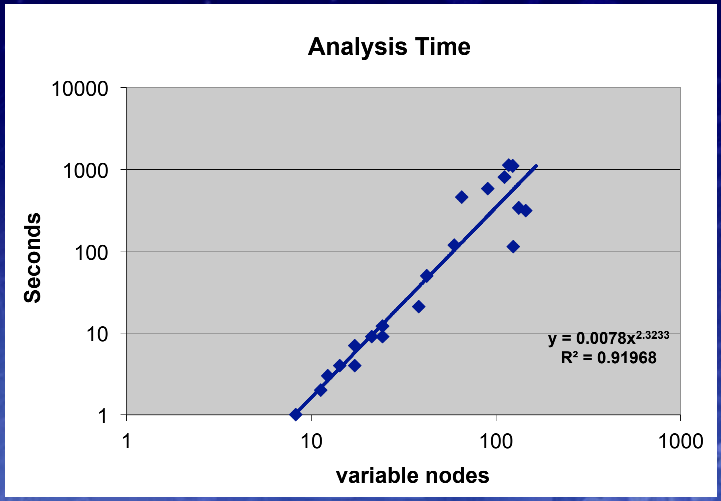
- Declarative logic programming language designed for databases
  - Horn clauses
  - Operates on relations
- Datalog is expressive
  - Relational algebra:
    - Explicitly specify relational join, project, rename
  - Relational calculus:
    - Specify relations between variables; operations are implicit
  - Datalog:
    - Allows recursively-defined relations

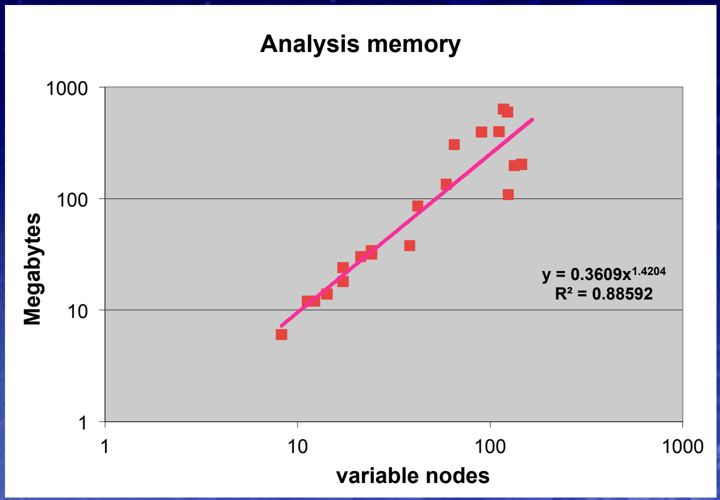
Datalog → BDD

- Join, project, rename are directly mapped to built-in BDD operations
- Automatically optimizes:
  - Rule application order
  - Incrementalization
  - Variable ordering
  - BDD parameter tuning
  - Many more…

Experimental Results

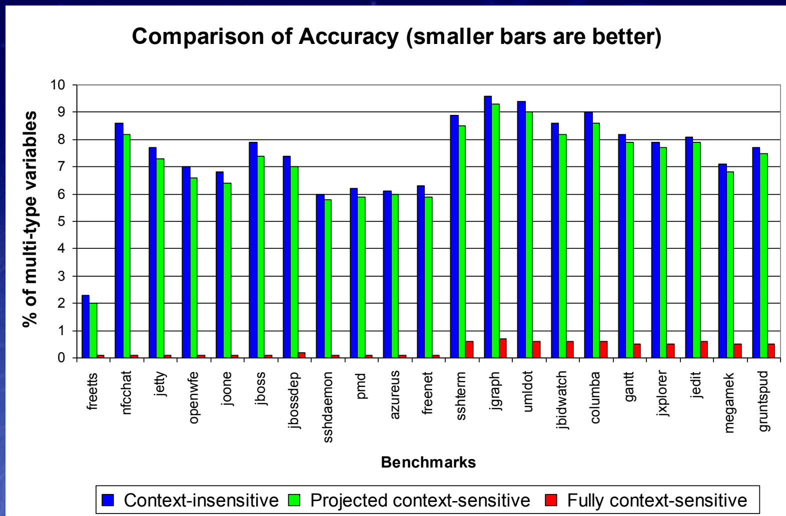
- Top 20 Java projects on SourceForge
  - Real programs with 100K+ users each
- Using automatic bddbddb solver
  - Each analysis only a few lines of code
  - Easy to try new algorithms, new queries
- Test system:
  - Pentium 4 2.2GHz, 1GB RAM
  - RedHat Fedora Core 1, JDK 1.4.2_04, javabdd library, Joeq compiler





Multi-type variables

- A variable is multi-type if it can point to objects of different types
  - Measure of analysis precision
  - One line in Datalog
- Two ways of handling context sensitivity:
  - Projected: Merge all contexts together
  - Full: Keep each context separate



Conclusion

- The first scalable context-sensitive inclusion-based pointer analysis
  – Achieves context sensitivity by cloning
- bddbddb: Datalog → efficient BDD
- Easy to query results, develop new analyses
- Very efficient!
  – < 19 minutes, < 600mb on largest benchmark
- Complete system is publicly available at: http://suif.stanford.edu/bddbddb