CS 243

Lecture 11

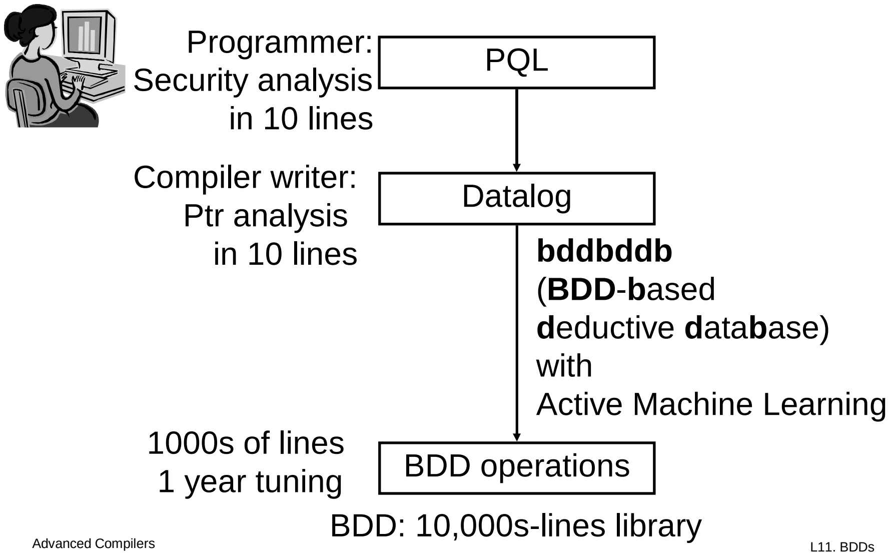
Binary Decision Diagrams (BDDs) in Pointer Analysis

1. Datalog → BDD
2. BDDs
3. Context-Sensitive Pointer Analysis
4. Performance of BDD Algorithms

Readings: Chapter 12

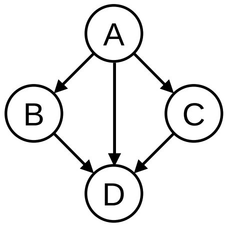
M. Lam & J. Whaley

Automatic Analysis Generation



1. Datalog → BDDs

- Example

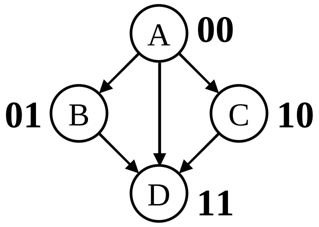


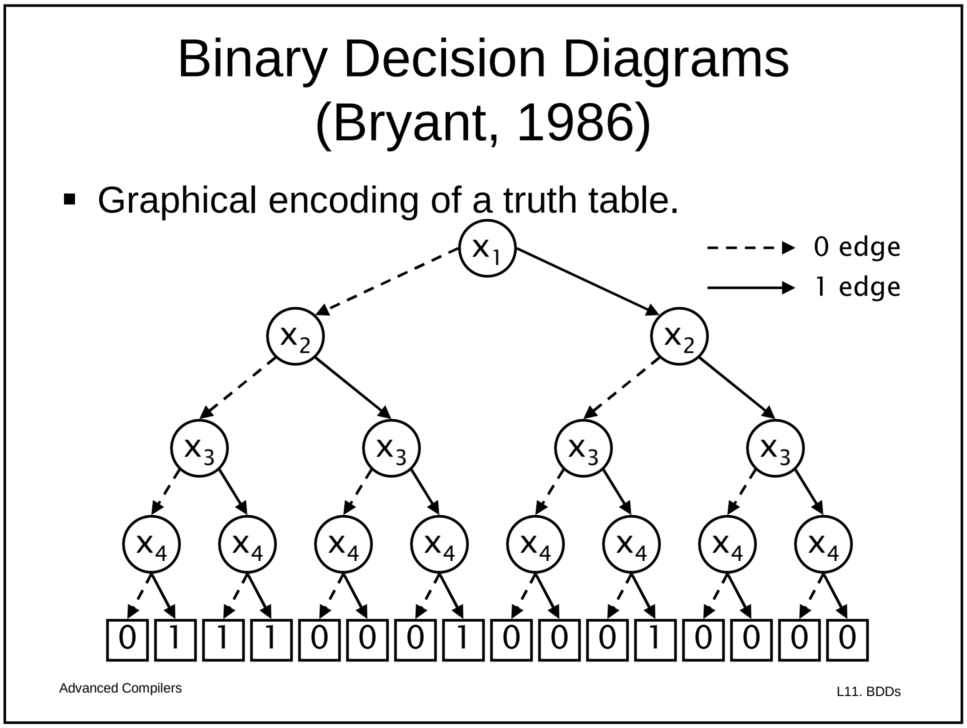
calls(A,B)
calls(A,C)
calls(A,D)
calls(B,D)
calls(C,D)

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



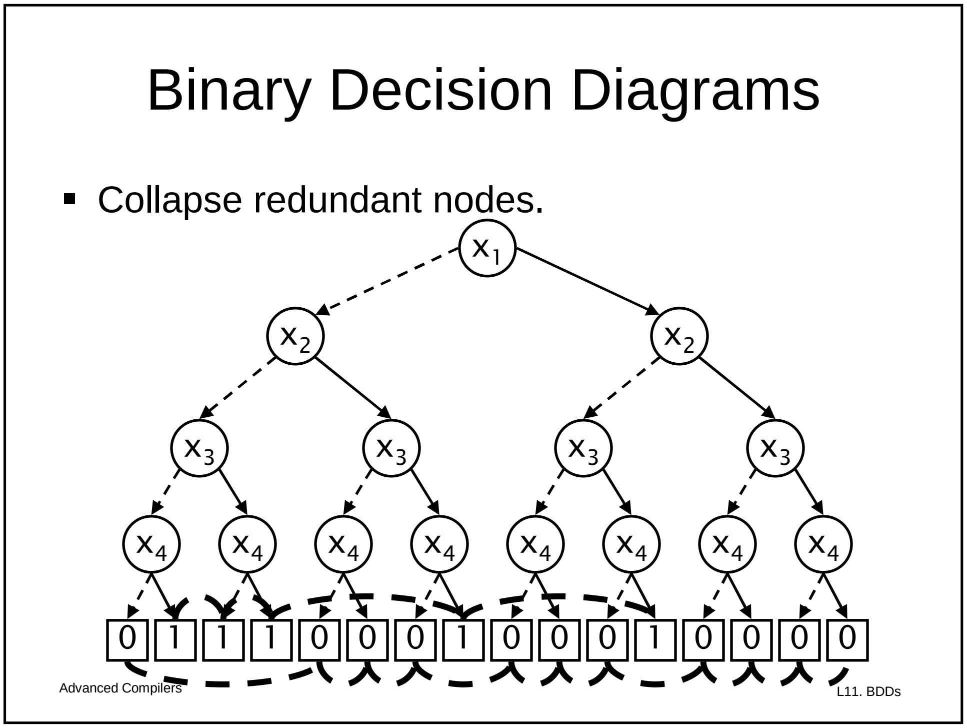


Binary Decision Diagrams (Bryant, 1986)

- Graphical encoding of a truth table.

Advanced Compilers

L11. BDDs



Binary Decision Diagrams

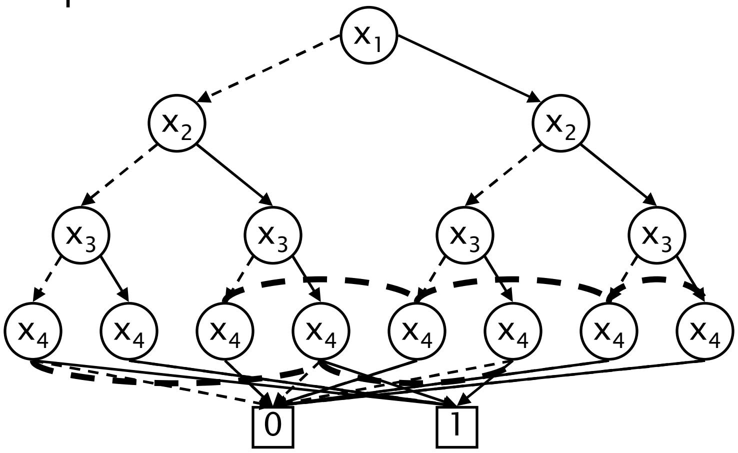
- Collapse redundant nodes.

Advanced Compilers

L11. BDDs

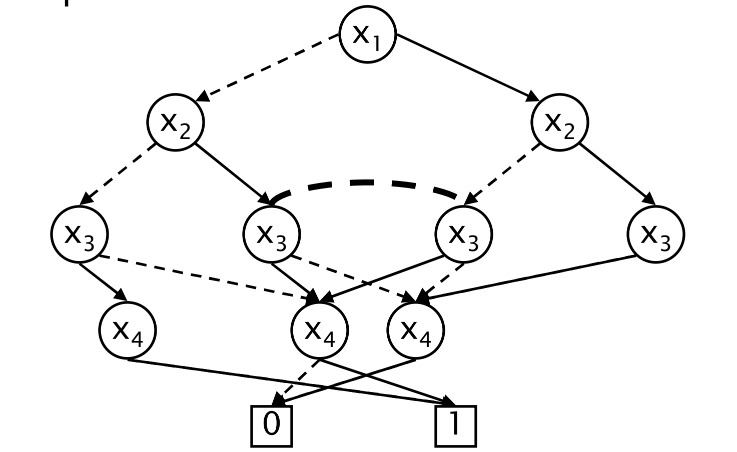
Binary Decision Diagrams

- Collapse redundant nodes.



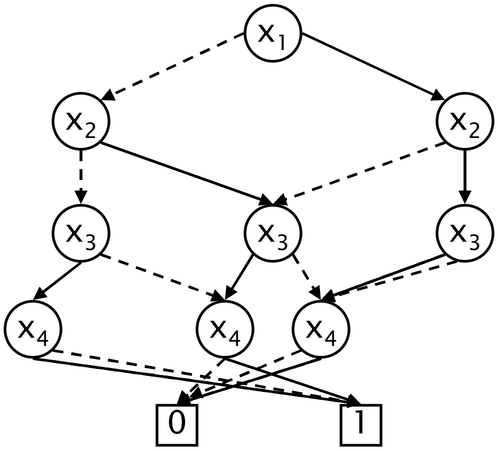
Binary Decision Diagrams

- Collapse redundant nodes.



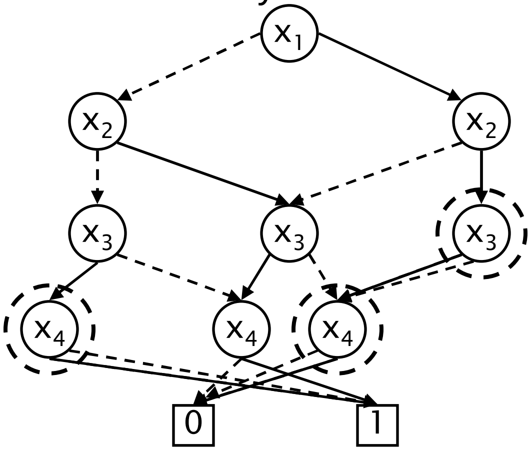
Binary Decision Diagrams

- Collapse redundant nodes.



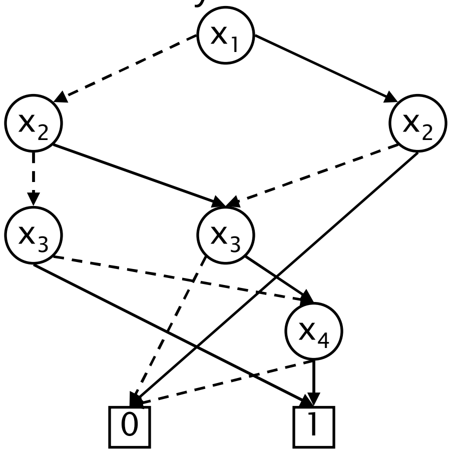
Binary Decision Diagrams

- Eliminate unnecessary nodes.



Binary Decision Diagrams

- Eliminate unnecessary nodes.



Datalog → BDDs

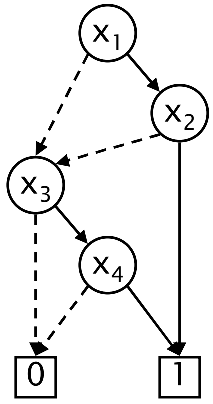
Datalog	BDDs
Relations	Boolean functions
Relation ops: ⋈, ∪, select, project	Boolean function ops: ∧, ∨, −, ~
Relation at a time	Function at a time
Semi-naïve evaluation	Incrementalization
Fixed-point	Iterate until stable

2. Binary Decision Diagrams

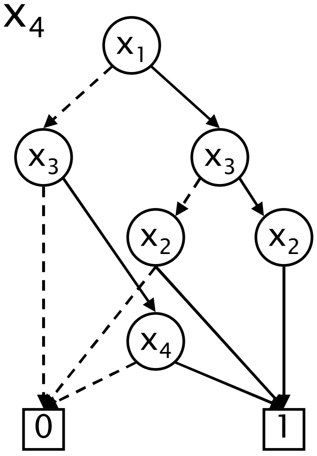
- Represent tiny and huge relations compactly
- Size depends on redundancy
  - Similar contexts have similar numberings
  - Variable ordering in BDDs

BDD Variable Order is Important!

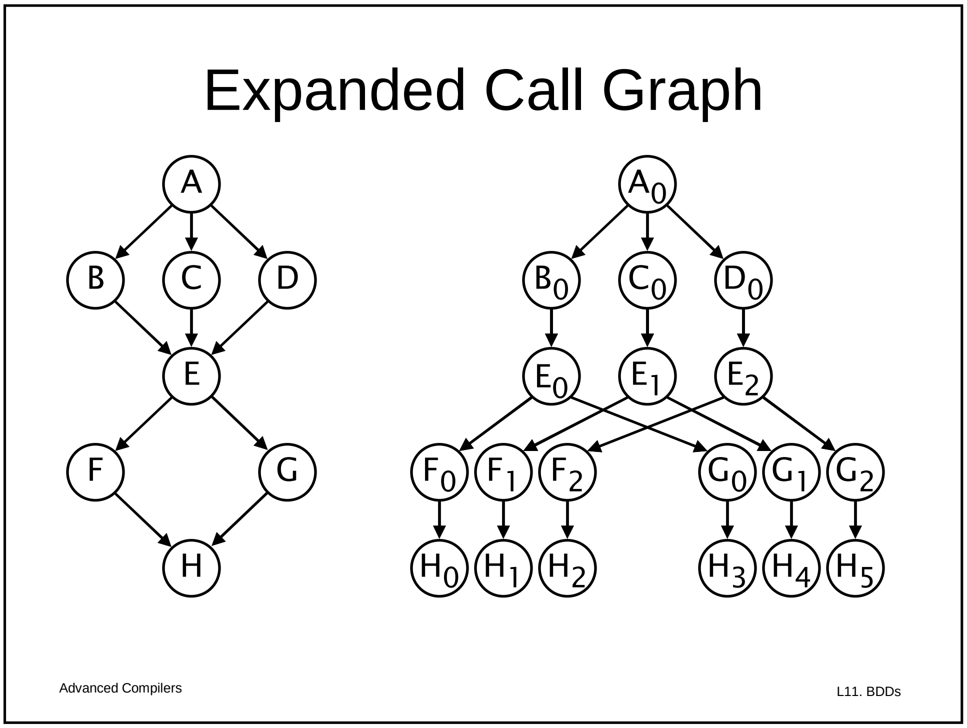
$x_1x_2 + x_3x_4$



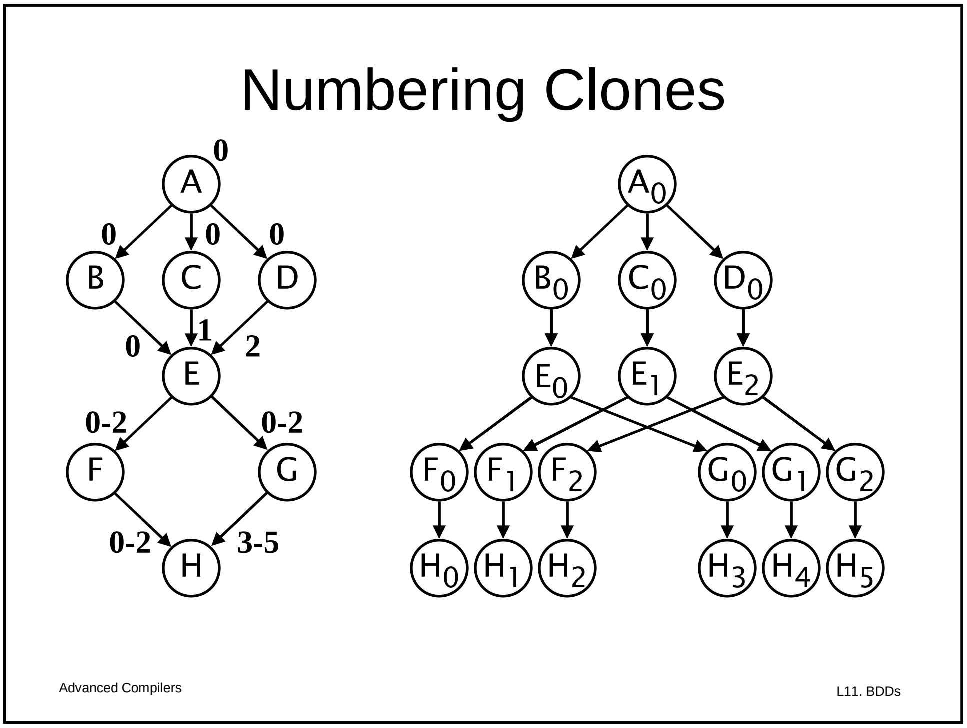
$x_1 < x_2 < x_3 < x_4$



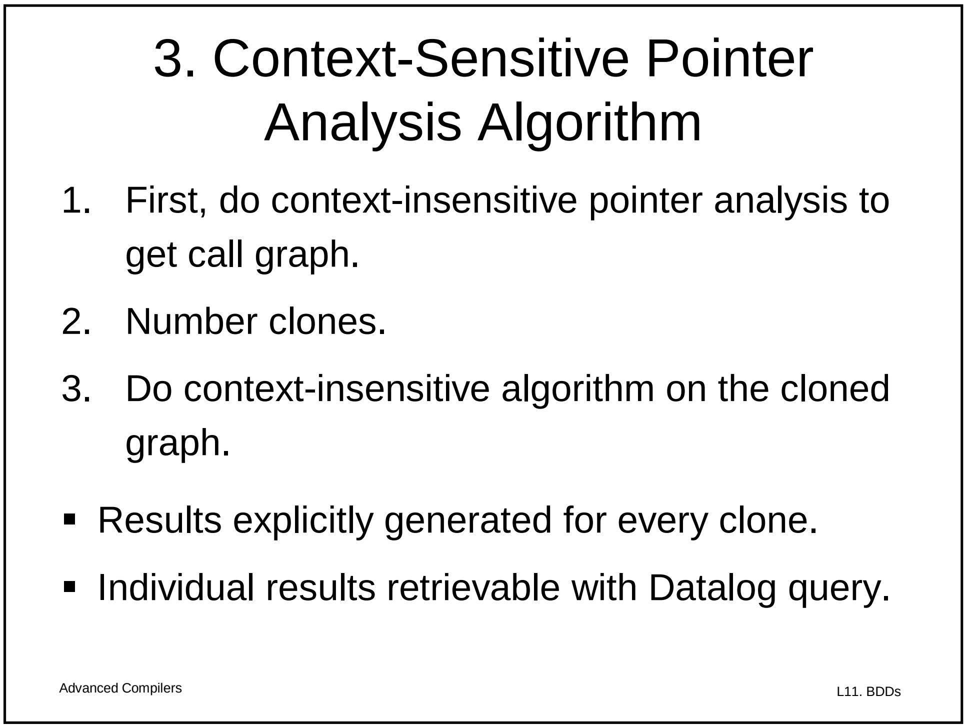
$x_1 < x_3 < x_2 < x_4$



Expanded Call Graph



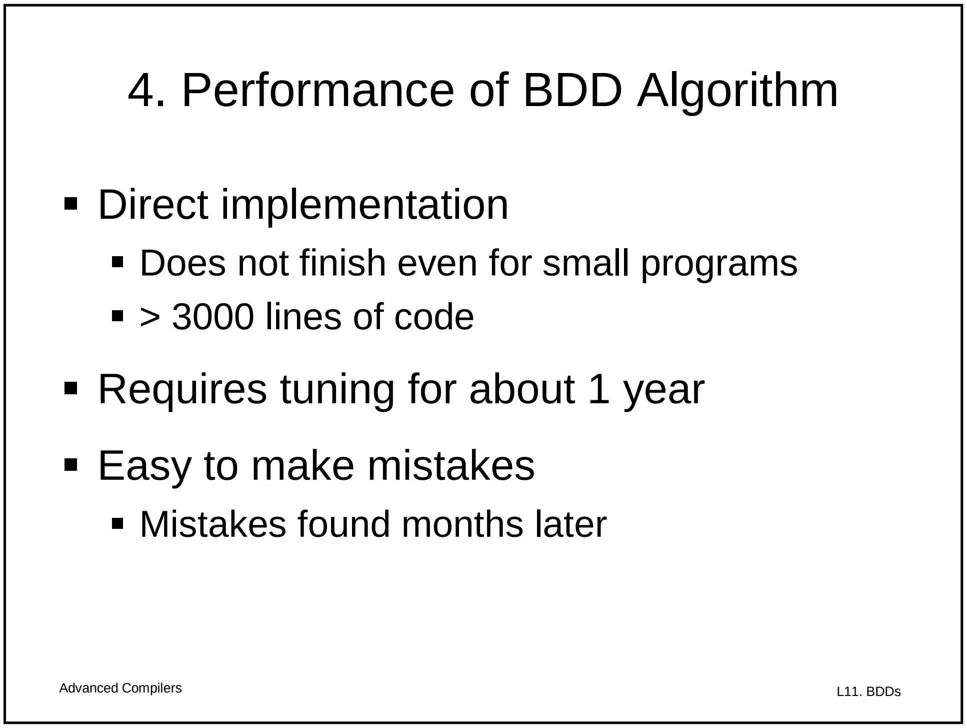
Numbering Clones



3. 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.



4. Performance of BDD Algorithm

- 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*

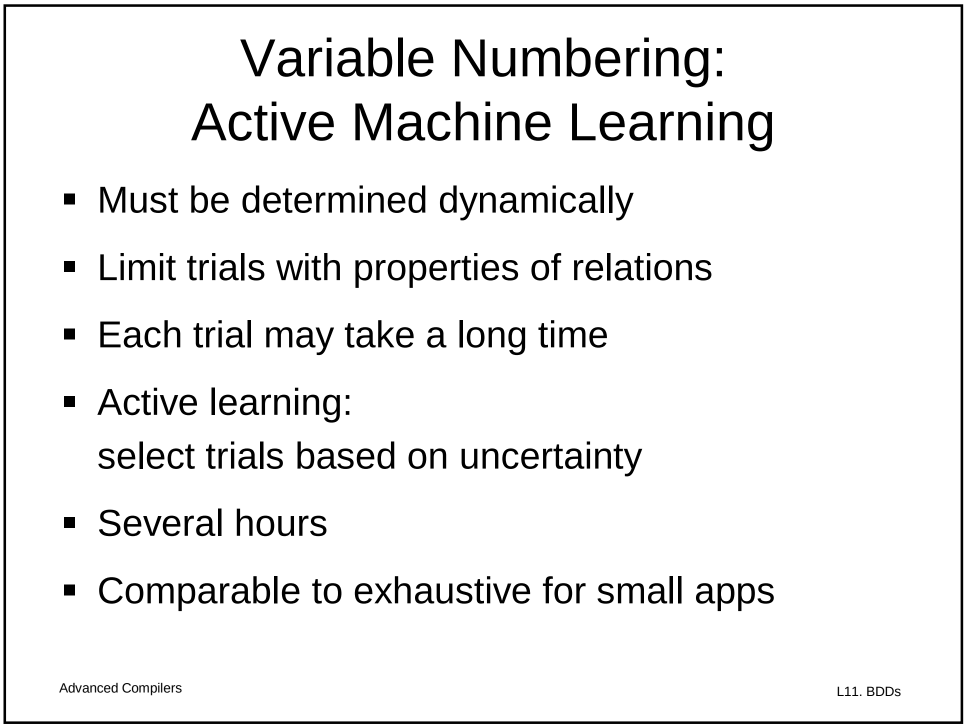
An Adventure in BDDs

- 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

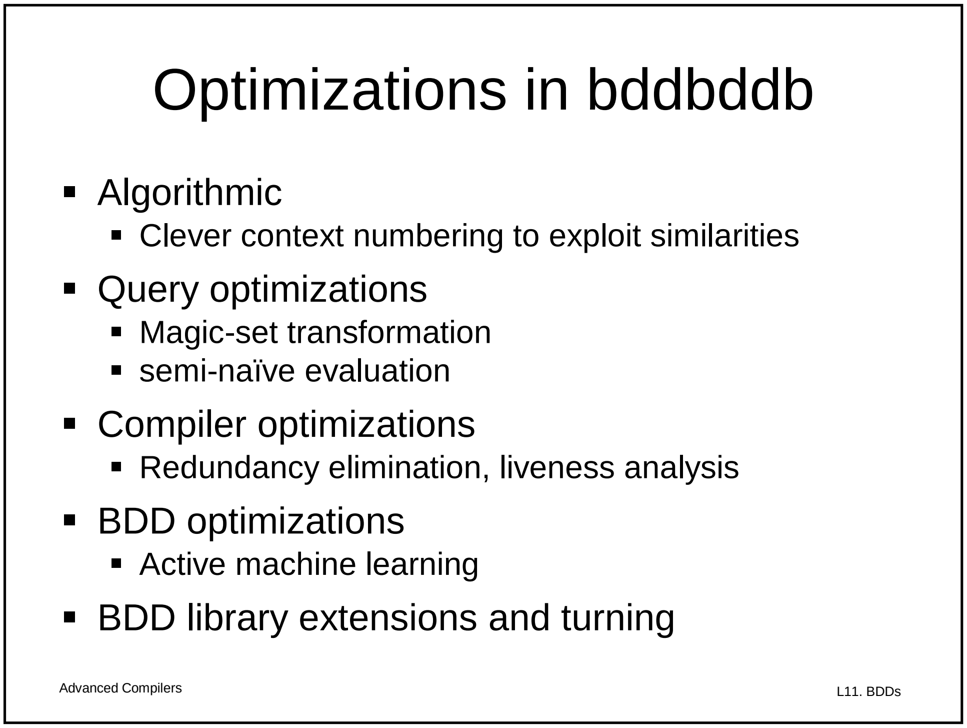
- Compacted BDD nodes
  - 20 bytes → 16 bytes *Time: 38m*
- Improved BDD hashing function
  - Simpler hash function. *Time: 37m*

- Total development time: 1 year
  - 1 year per analysis?!?
- Optimizations obscured the algorithm.
- Many bugs discovered, maybe still more.



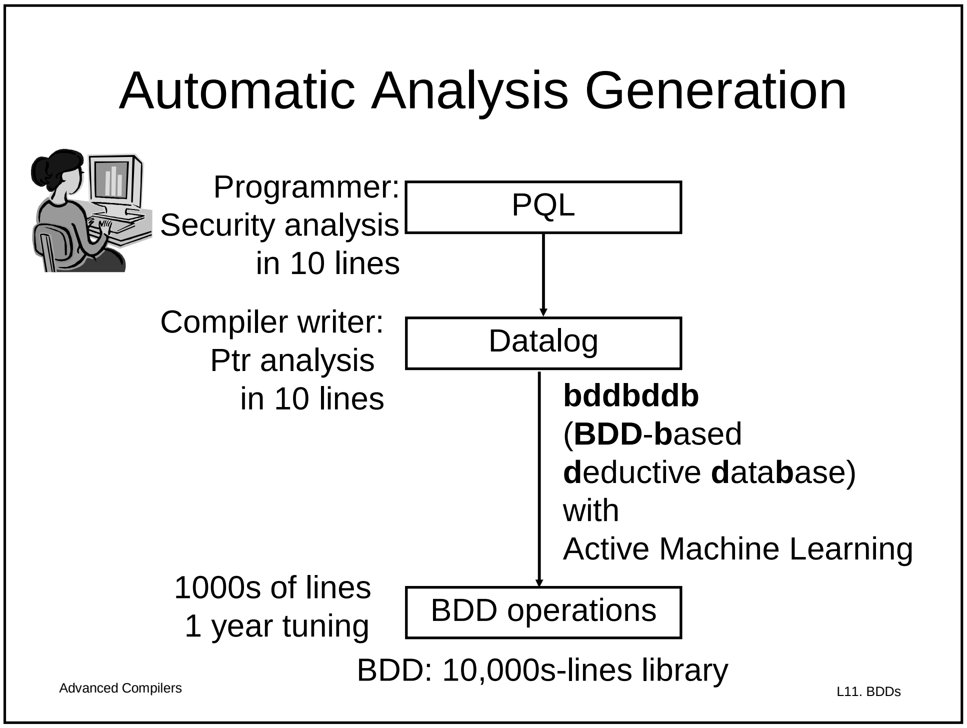
Variable Numbering: Active Machine Learning

- 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



Optimizations in bddbddb

- Algorithmic
  - Clever context numbering to exploit similarities
- Query optimizations
  - Magic-set transformation
  - semi-naïve evaluation
- Compiler optimizations
  - Redundancy elimination, liveness analysis
- BDD optimizations
  - Active machine learning
- BDD library extensions and turning



Automatic Analysis Generation

Programmer:
Security analysis
in 10 lines

PQL

Compiler writer:
Ptr analysis
in 10 lines

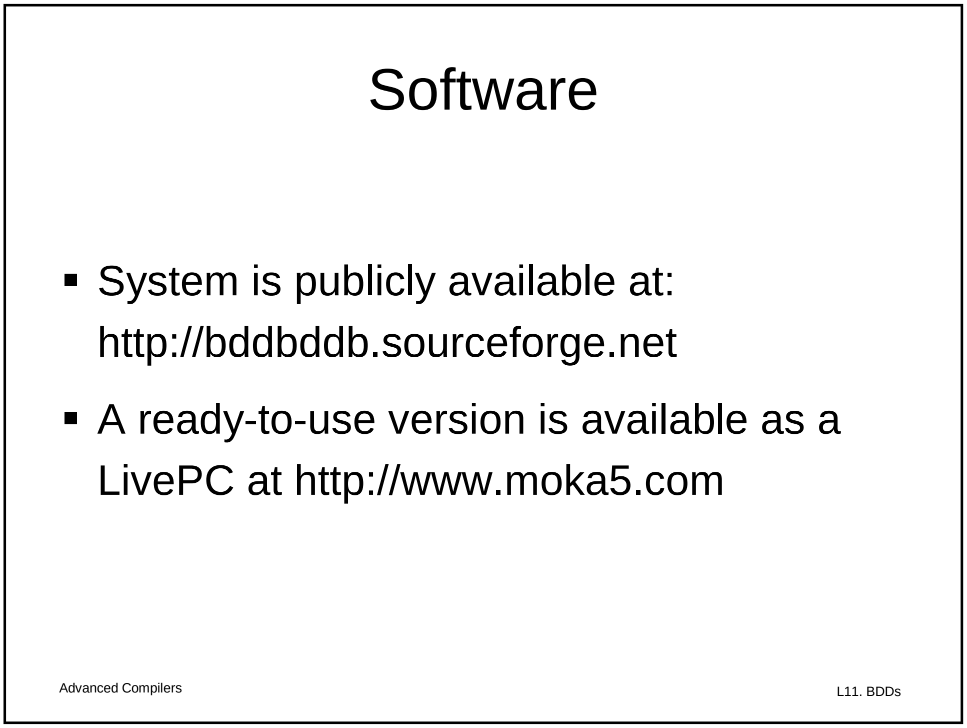
Datalog

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



Software

- System is publicly available at: http://bddbddb.sourceforge.net
- A ready-to-use version is available as a LivePC at http://www.moka5.com