Accurate and Efficient Predicate Analysis with Binary Decision Diagrams

John W. Sias Wen-mei W. Hwu David I. August

IMPACT Compiler Research Group
Center for Reliable and High-Performance Computing
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

Liberty Computer Architecture
Research Group
Department of Computer Science
Princeton University

33rd Annual ACM/IEEE International Symposium on Microarchitecture

IMPACT
ADVANCED COMPILER TECHNOLOGY

Predicated compilation

- New responsibilities, opportunities for the EPIC compiler
- Multiple active paths of control through predication
  - Keeping optimizations effective
  - Managing new scheduling, register allocation problems
  - New form for explicit speculation
  - Enabling new optimizations
- Predication entails much more than just removing branches
  - Predicate optimization [Aug99]
  - Flexible control flow expression [AHM97]
  - Integrated optimization and scheduling models [Car99][EMM00]
  - Predicate-aware register allocation [ED95][Gil96]

Predicated compilation example

Source code

```
if (x>-8&&x<8) {
  stmtA;
  if (x == 0)
    stmtB;
}
else {
  stmtC;
  if (x < 0)
    stmtD;
}
```

If-converted

```
     p2_uf = (0=0)
     ----------------------
     p1_ut,p2_of = (r1>-8)
     ----------------------
(p1) p3_ut,p2_of = (r1<8)
     ----------------------
(p3) p4_ut = (r1=0)
(p2) p5_ut = (r1<0)
(p3) stmtA
(p2) stmtC
     ----------------------
(p4) stmtB
(p5) stmtD
```

Optimized

```
     p3_ut,p2_uf = (0=0)
     ----------------------
     p3_at,p2_of = (r1>-8)
     p3_at,p2_of = (r1<8)
     p4_ut = (r1=0)
     p5_ut = (r1<=-8)
     ----------------------
(p3) stmtA
(p4) stmtB
(p2) stmtC
(p5) stmtD
```

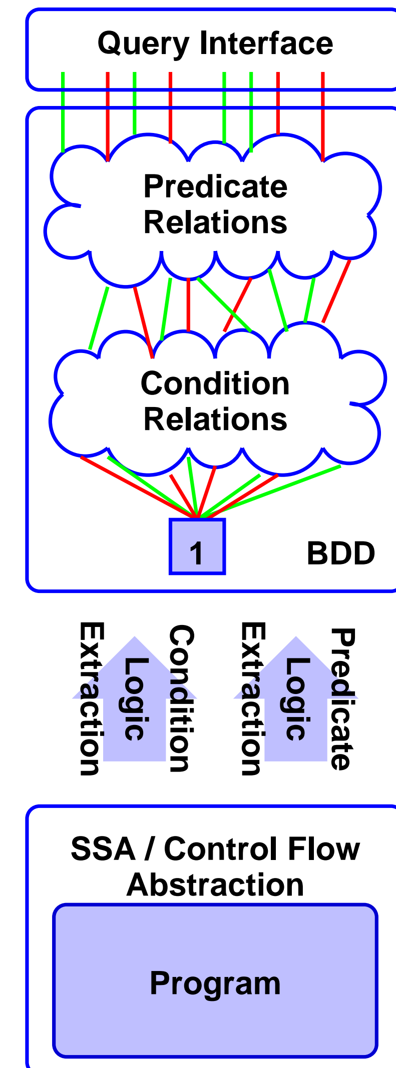
- *If-conversion* changes branches to predication
- Predicate define semantics: *unconditional*, *or*, *and*
- Scheduling reorders predicated operations
- Predicate optimization flattens predicate expressions [Aug99]
  - Guard predicate removal
  - Condition knowledge required for p4 and p5.

Why do we need another predicate analysis?

- Predicate Hierarchy Graph [Mah92]
  - Genealogical predicate relations
  - Lack of canonicity limits accuracy
- P-facts [ED95]
  - Extracts invariants from conditions, predicates
  - General expression solver
- Predicate Query System (PQS) [JS96][Gil96]
  - Graph based on partitions found in if-converted code
  - Expression-based dataflow model
- Non-if-conversion use of predication
- New compilation models
- Need for high accuracy, flexible feature integration, production-ready algorithms

The Predicate Analysis System

- Objectives:
  - Efficient representation and query
  - Integrated analysis of predicates and conditions
  - Support general use of predication
- Map program logic onto Boolean substrate
  - Predicate semantics
  - Condition relations
  - Control flow



IMPACT ADVANCED COMPILER TECHNOLOGY

Reduced Ordered Binary Decision Diagrams [BRB90]

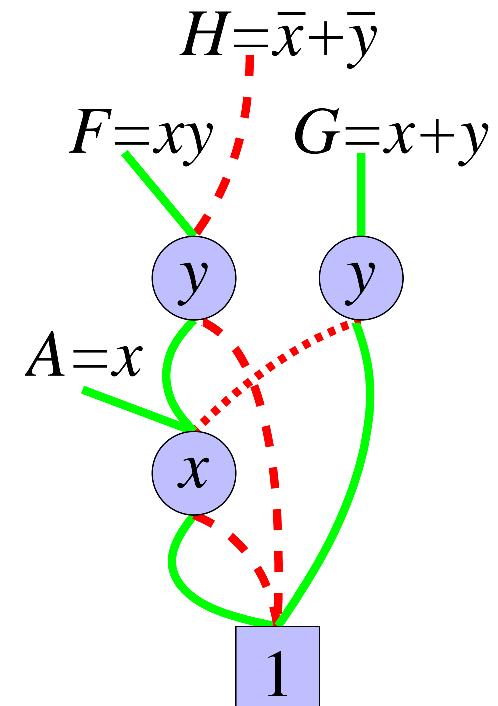
- Canonical, given a variable order
  - Unique “formula” for each function
  - Optimal reuse of subexpressions
  - Memoization
- $G = \text{ITE}(F, T, E)$ or $\{F, T, E\}$

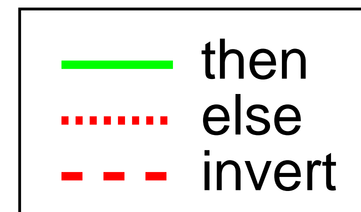
  $= \text{ITE}(v, \{F_v, T_v, E_v\}, \{F_{\overline{v}}, T_{\overline{v}}, E_{\overline{v}}\})$
- Example: $I = FG \Rightarrow I = \text{ITE}(F, G, 0)$

  $I = \text{ITE}(y, \{F_y, G_y, 0\}, \{F_{\overline{y}}, G_{\overline{y}}, 0\})$

  $I = \text{ITE}(y, \{A, 1, 0\}, \{0, A, 0\})$

  $I = \text{ITE}(y, A, 0) = F$
- Formulae must be defined at time of use

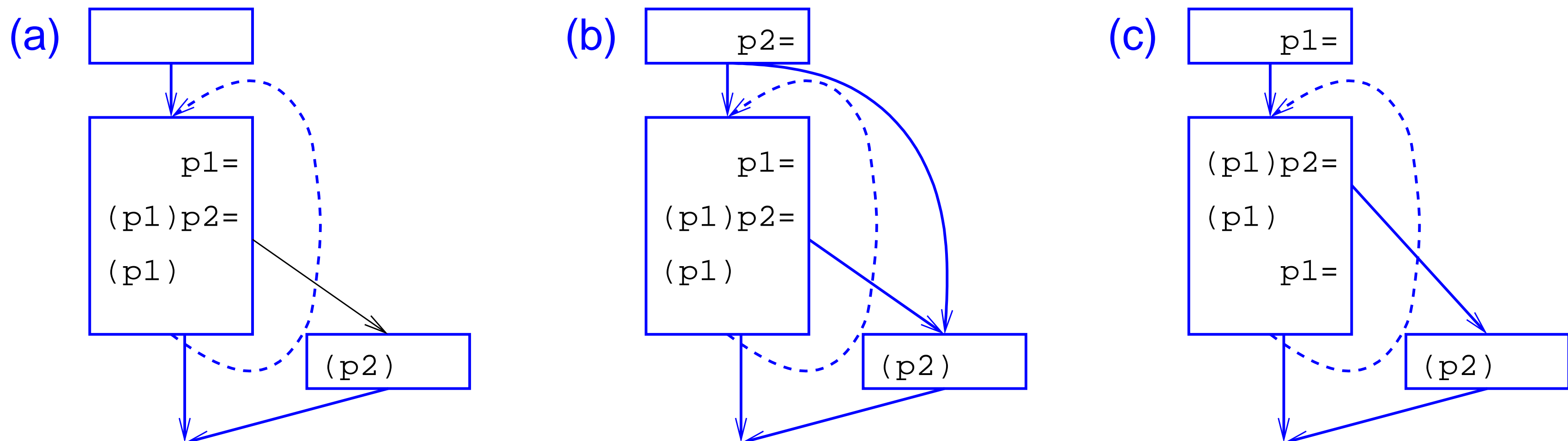




IMPACT ADVANCED COMPILER TECHNOLOGY

Control flow abstraction

- Relevant reaching definitions must be uniquely identified
- Traverse predicate definitions in topological order
- Three levels of support for predicate liveness:

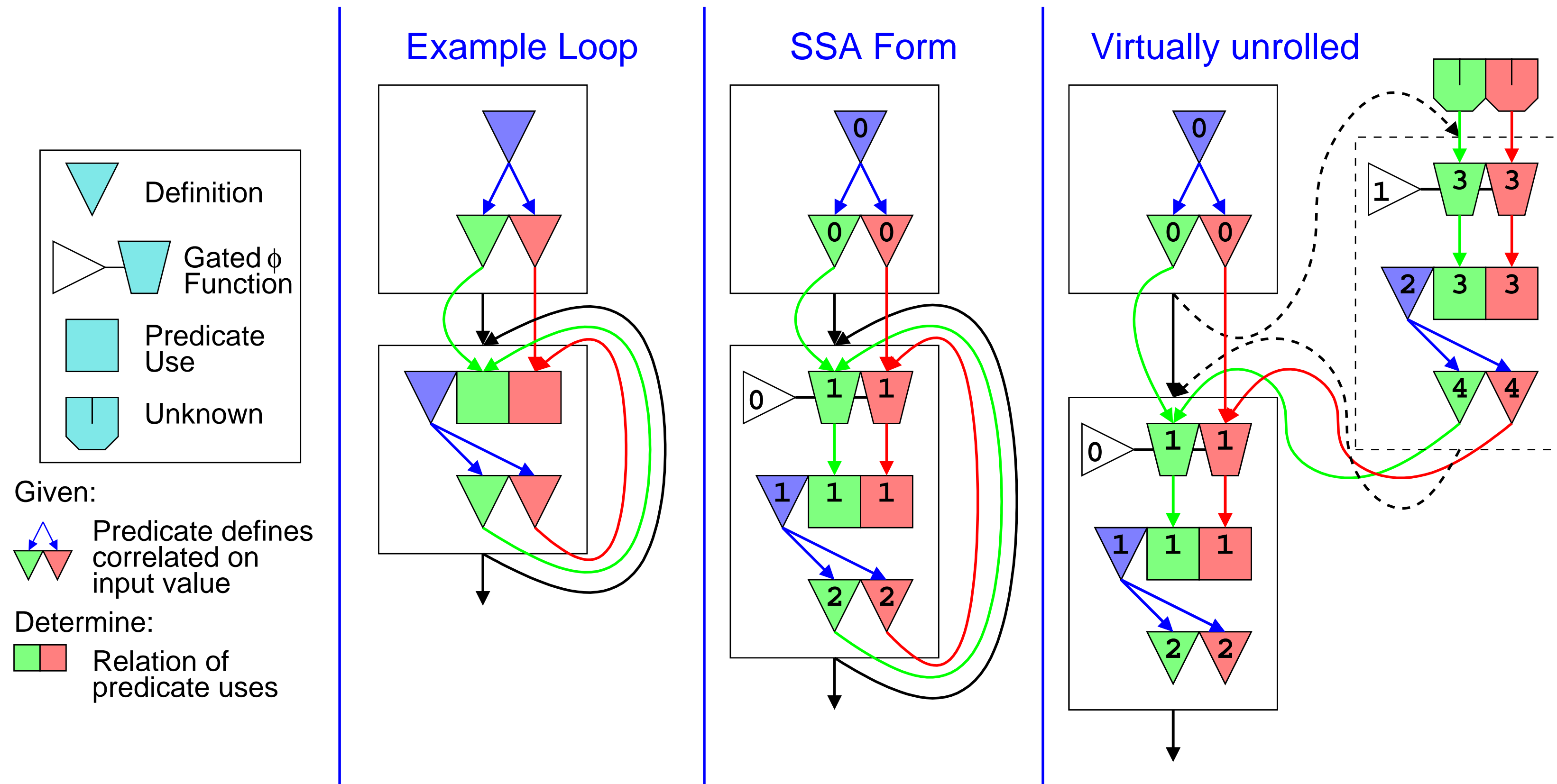


(a) Forward flow, single definition: ϕ-less SSA

(b) Forward flow, multiple definition: GSA

(c) General (cyclic) flow: GSA, virtual unrolling

Cyclic control flow and virtual unrolling

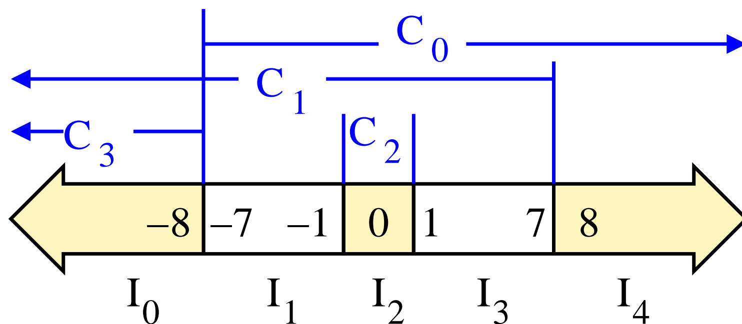


- "Virtually unroll" loops with new SSA names
- Provides one application of loop "filter" to unknowns
- Use gating to correlate values

Mapping condition families to Boolean space

- Condition family: set of logically related comparisons
  - Member values uniquely determined given an “outcome”
  - Mutually exclusive, collectively exhaustive outcomes
- *Finite domain*: use $\lceil \log_2 n \rceil$ variables to represent n outcomes

```
p1.0_ut,p2.0_uf=(0=0)
p1.1_at,p2.1_of=(r1>-8)    C0
p1.2_at,p2.2_of=(r1<8)     C1
        p4.0_ut=(r1=0)     C2
        p5.0_ut=(r1<=-8)   C3
```

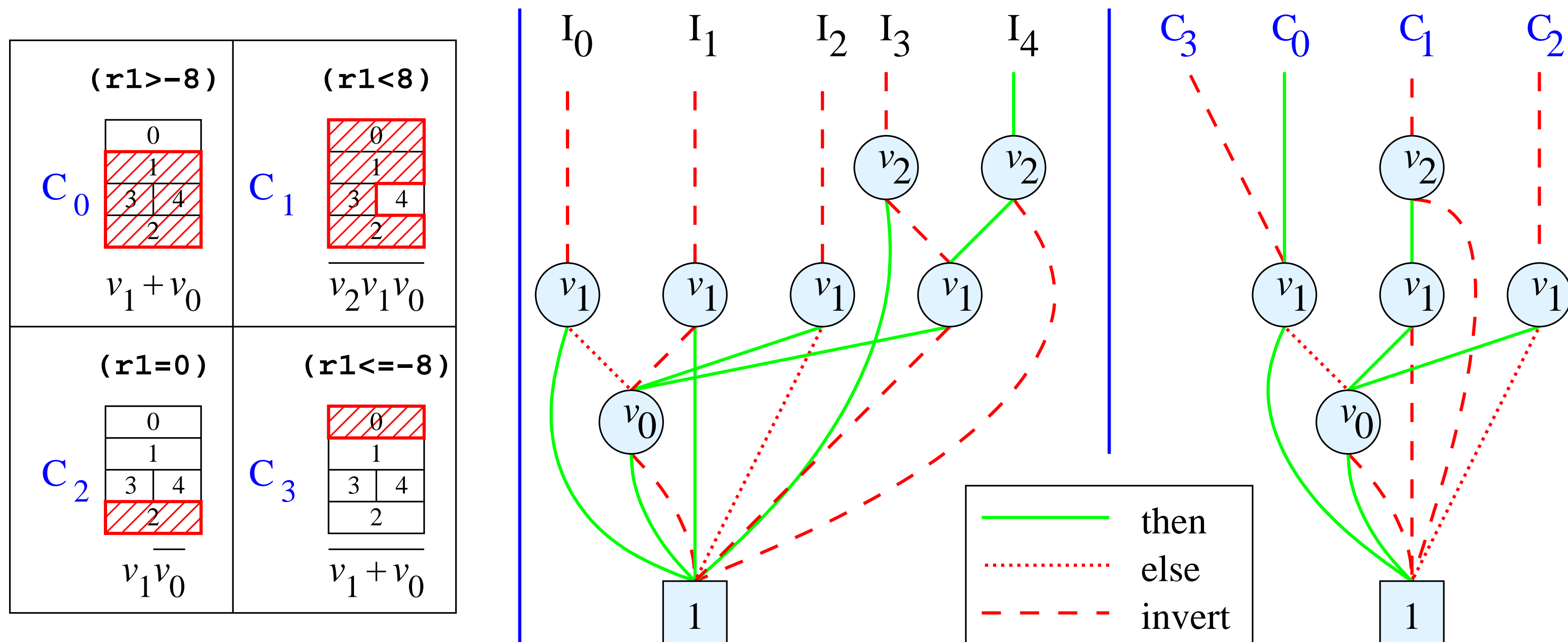


v_1v_0 \ v_2	0	1
00	I_0	
01	I_1	
11	I_3	I_4
10	I_2	

C_0: $v_1 + v_0$	C_1: $\overline{v_2 v_1 v_0}$
C_2: $v_1 \overline{v_0}$	C_3: $\overline{v_1 + v_0}$

Mapping condition families to BDD

- *Finite domain*: use $\lceil \log_2 n \rceil$ variables to represent n outcomes
- Condition formulae form basis for predicate graph

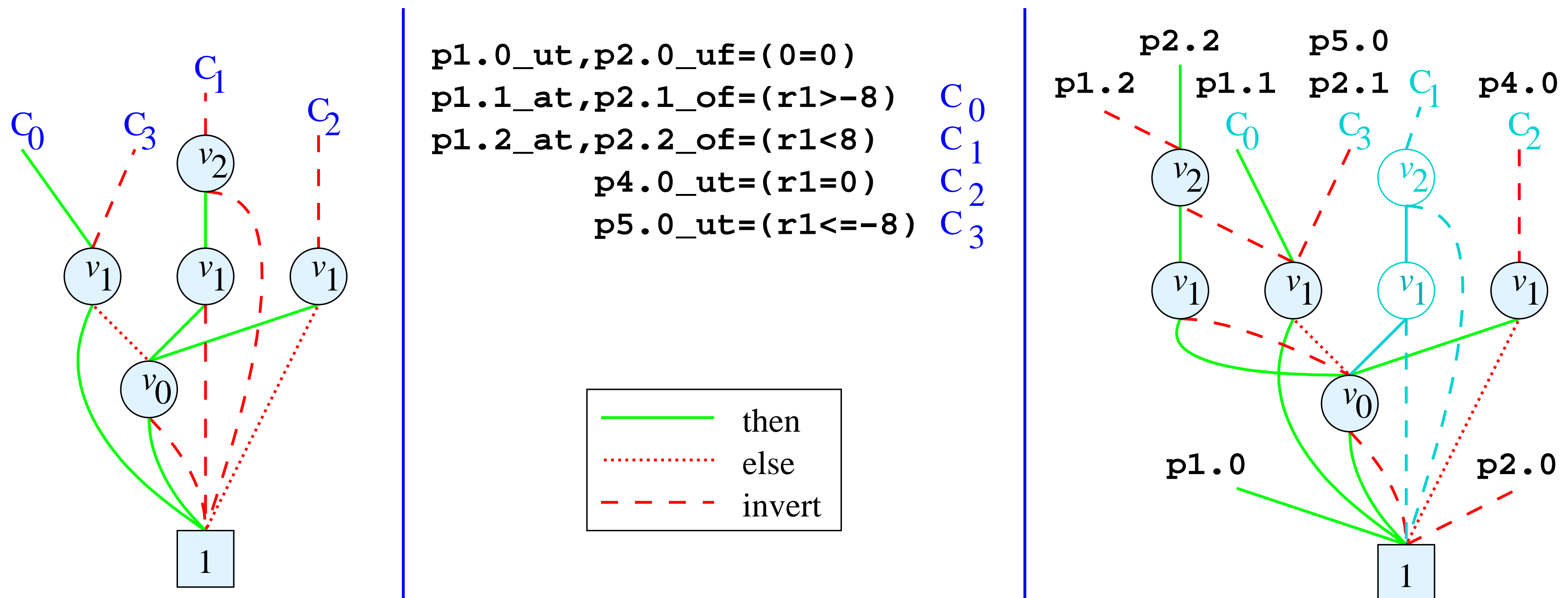


- BDD garbage collects nodes left from outcome enumeration

IMPACT ADVANCED COMPILER TECHNOLOGY

Mapping predicate define semantics

- Each predicate deposit type maps to an ITE formula, *e.g.*
  - $(\mathtt{p}g)\ \mathtt{p}i.j_\mathtt{ut} = C \Rightarrow \mathtt{p}_{i.j} = \mathrm{ITE}(C, \mathtt{p}_g, 0)$
  - $(\mathtt{p}g)\ \mathtt{p}i.j_\mathtt{at} = C \Rightarrow \mathtt{p}_{i.j} = \mathrm{ITE}(\mathtt{p}_g, \mathrm{ITE}(C, \mathtt{p}_{i.j-1}, 0), \mathtt{p}_{i.j-1})$

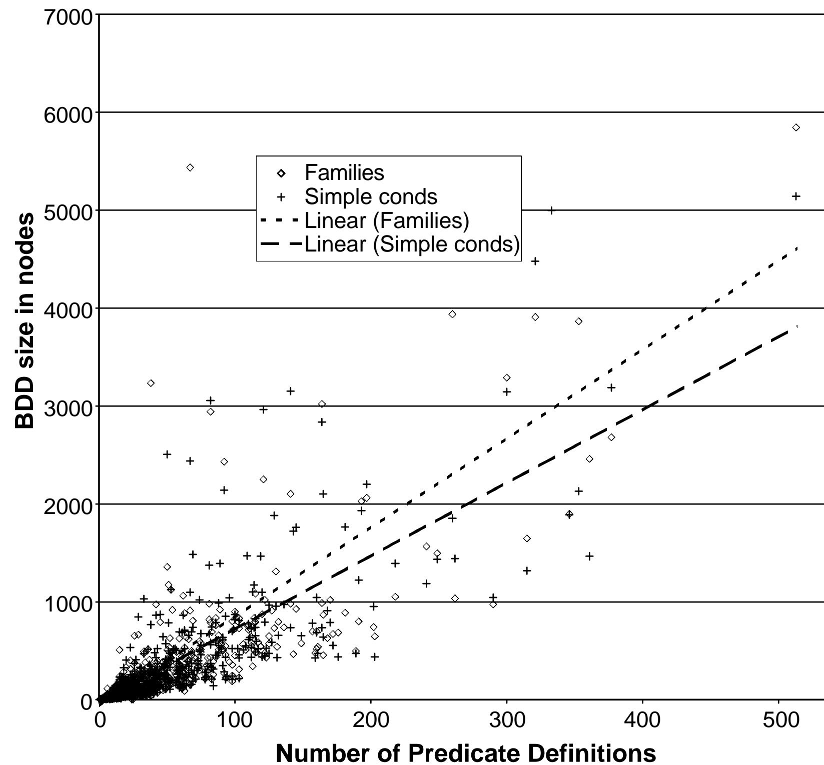


- Queries also by ITE: subset, intersection, union-subsumes

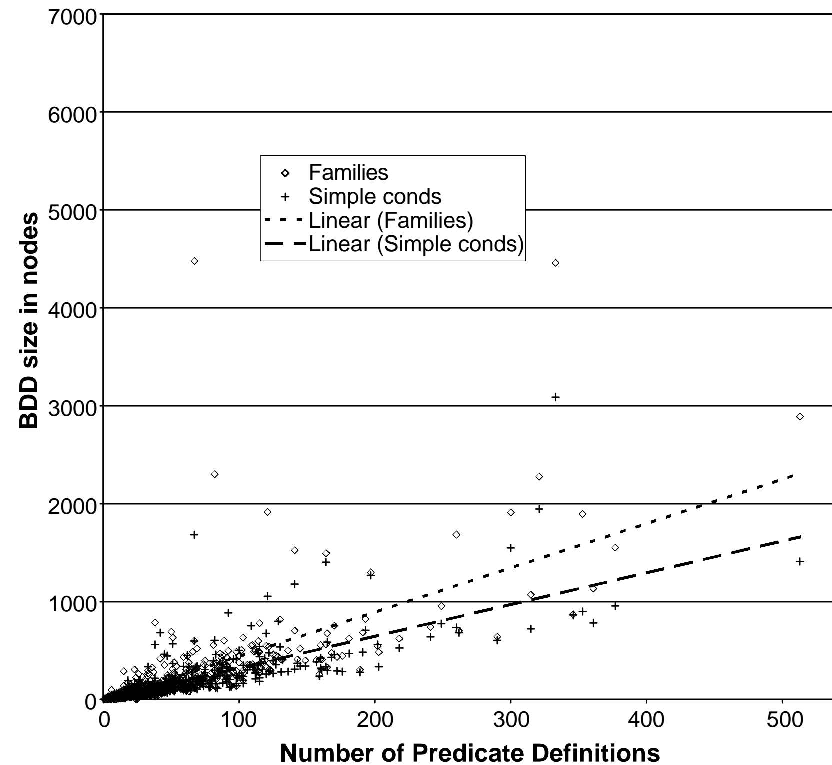
Size of the representation

- All Boolean function representations can “explode”
  - Explosion requires $\propto 2^{\text{number of variables}}$
  - In BDD form, this equates to a full binary tree
  - Circuit example for BDD: integer multiplier
- “Intermediate size” of function determines BDD size
  - $x_1x_2 + x_3x_4 + ... + x_{2n-1}x_{2n}$
  - $x_1x_{n+1} + x_2x_{n+2} + ... + x_nx_{2n}$
  - b bits necessary to represent “live variables”
  - BDD encodes all possible paths $\Rightarrow$ size $\propto 2^b$
- Variable ordering key to controlled growth

Predicate BDD growth



Original order



After variable sifting

- Ordering “naturally” delivers reasonable size
- Potential exponential growth can be short-circuited
- Semantic mapping achieved an efficient representation

IMPACT ADVANCED COMPILER TECHNOLOGY

For further information

- IMPACT publications and compiler framework: `http://www.crhc.uiuc.edu/IMPACT/`
- [Som98] F. Somenzi, "CUDD, Colorado University Decision Diagrams," http://vlsi.colorado.edu/∼fabio/CUDD/, 1998.
- [Mah92] S. Mahlke *et al.*, "Effective compiler support for predicated execution," Proc. MICRO-25, pp. 45-54, Dec. 1992.
- [BRB90] K. Brace, R. Rudell, and R. Bryant, "Efficient implementation of a BDD package," Proc. IEEEDAC-27, pp. 40-45, Jan. 1990.
- [Gil96] D. Gillies, D. Ju, and M. Schlansker, "Global predicate analysis and its application to register allocation," Proc. MICRO-29, pp. 114-125, Dec. 1996.
- [Aug99] D. August *et al.*, "The program decision logic approach to predicated execution," Proc. ISCA-26, pp. 208-219, May 1999.