Binary Decision Diagrams

Gianpiero Cabodi ***Stefano Quer***

Politecnico di Torino

Torino, Italy

{gianpiero.cabodi,stefano.quer}@polito.it

http://staff.polito.it/{gianpiero.cabodi,stefano.quer}/

Reference

- **Paper**
  - **R. E. Bryant**
    "Graph-based Algorithms for Boolean Function Manipulation"
    IEEE Transaction on Computers,
    Vol. C-35, No. 8, August 1986, pp. 677-691
    (most cited CS paper !!!)
- **Books**
  - **C. Meinel, T. Theobald**
    "Algorithms and Data Structure in VLSI Design"
    Springer-Verlag, Berlin, August 1998
    ISBN 3-540-64486-5
  - **G. D. Hachtel, F. Somenzi**
    "Loginc Synthesis and Verification Algorithms"
    Kluwer Academic Publishers

Outline

- **Binary Decision Diagrams: Fundamentals**
- **Generation of BDDs from Network**
- **Variable Ordering Related Problems**
- **Complex Operations with BDDs**
- **Some Conclusions on BDDs**
- **BDD Packages**

Outline

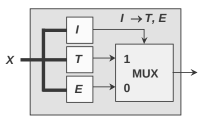
- **Binary Decision Diagrams: Fundamentals**
- **Generation of BDDs from Network**
- **Variable Ordering Related Problems**
- **Complex Operations with BDDs**
- **Some Conclusions on BDDs**
- **BDD Packages**

Binary Decision Diagrams

- **Restricted Form of Branching Program**
  (graph representation of Boolean function)
- **Canonical form (constant time comparison)**
- **Simple (Polynomial) algorithms to construct e manipulate (Boolean operations: and, or, not, etc.)**
- **Exponential but practically efficient algorithm for boolean quantification**
- **Starting Point**
  1. **If-Then-Else Decomposition** — **Decomposition**
  2. **Ordered Decision Tree**
  3. **Reduced Decision Tree** — **Reduction**

If-Then-Else Decomposition

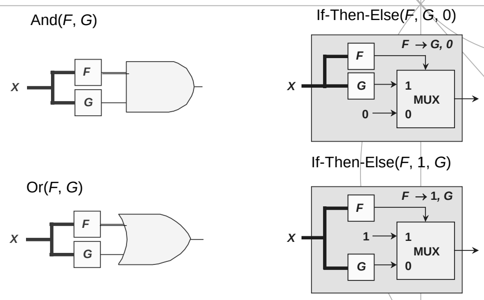
- **All operators can be expressed in terms of ITE**
- **Used to build BDD from logic network or formula**



Arguments *I*, *T*, *E*
- **Functions over variables** *X*
- **Represented as BDDs**

Result
- ITE (I, T, E) = $(I \wedge T) \vee (\neg I \wedge E)$
- **Represented as a BDD**

- **All operators can be expressed using ITE**



- **¬ x ➔ ITE (x, 0, 1)**
- **x == y ➔ ITE (x, ITE (y, 1, 0), ITE (y, 0, 1))**
- **...**

- **Boole's (Shannon) Decomposition**
  - **F ➔ ITE $(x, F|_x, F|_{\neg x})$**
  - **F = $(x \wedge F|_x) \vee (\neg x \wedge F|_{\neg x}) = x \cdot F|_{x=1} + \neg x \cdot F|_{x=0}$**
- **BDD from Boole's Decomposition**
  1. **Form decomposition one variable at a time**
  2. **Proceed till terminal (0-1) values**

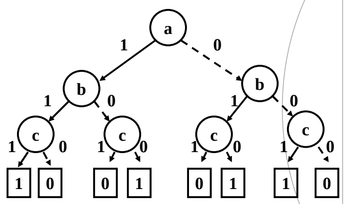
This gives an "Ordered Decision Tree"

Example 1

F (a,b,c) = (a ⊕ b) ⊕ c

Example 1

F (a,b,c) = (a ⊕ b) ⊕ c



Ordered Decision Tree

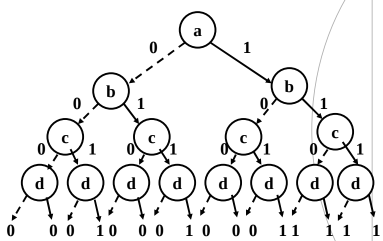
(complete tree ...
exponential size !!!)

Example 2

F (a, b, c, d) = (a∧b) ∨ (c∧d) = ab + cd
(order a, b, c, d)

Example 2

F (a, b, c, d) = (a∧b) ∨ (c∧d) = ab + cd
(order a, b, c, d)



Ordered Decision Tree

(complete tree ...
exponential size !!!)

Reduction Rules

1. **Combine isomorphic subtrees**
2. **Eliminate redundant nodes (those with identical children)**
3. **Use edge attributes (inverted edges)**
   (only one terminal nodes)

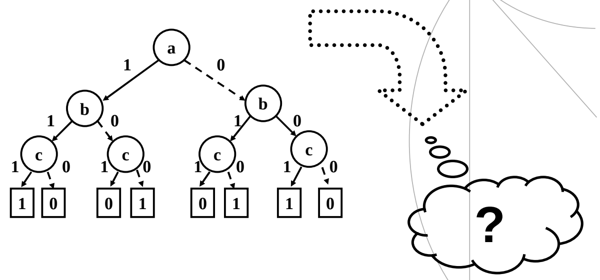
Then

- **Tree becomes a graph**
- **Ordered Decision Tree becomes BDD or ROBDD**
  1. **if the two children of a node are the same, the node is eliminated: f = vf + vf**
  2. **if two nodes have isomorphic graphs, they are replaced by one of them**

  These two rules make it so that each node represents a distinct logic function.

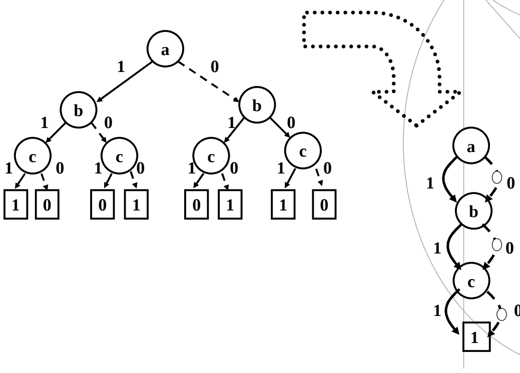
Example 1

F (a,b,c) = (a ⊕ b) ⊕ c



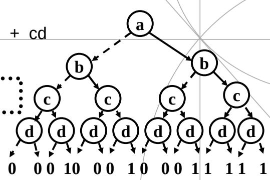
Example 1

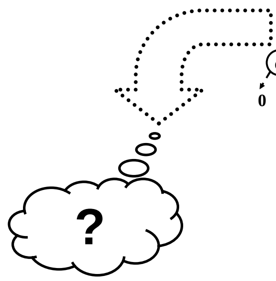
F (a,b,c) = (a ⊕ b) ⊕ c



Example 2

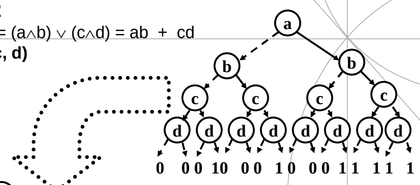
F (a, b, c, d) = (a∧b) ∨ (c∧d) = ab + cd
(order a, b, c, d)

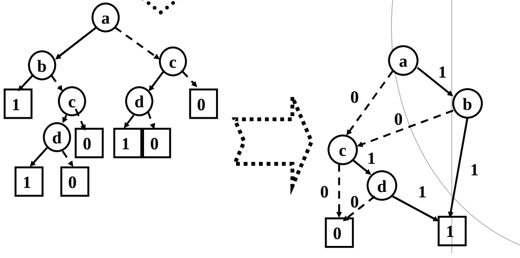




Example 2

F (a, b, c, d) = (a∧b) ∨ (c∧d) = ab + cd
(order a, b, c, d)





To sum up …

- **A BDD (ROBDD)**
  - **Is a directed acyclic graph (DAG)**
    - **one root node, two terminals 0, 1**
    - **each node, two children, and a variable**
  - **It uses a Shannon co-factoring tree, except that it is**
    - **Reduced**
    - **Ordered**
  - **Reduced**
    - **any node with two identical children is removed**
    - **two nodes with isomorphic BDD's are merged**
  - **Ordered**
    - **Co-factoring variables (splitting variables) always follow the same order along all paths**

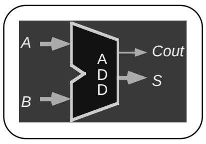
$$x_{i_1} < x_{i_2} < x_{i_3} < \ldots < x_{i_n}$$

Outline

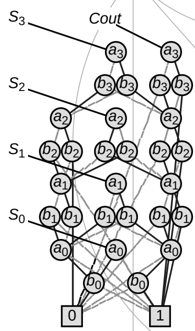
- ❖ Binary Decision Diagrams: Fundamentals
- ❖ Generation of BDDs from Network
- ❖ Variable Ordering Related Problems
- ❖ Complex Operations with BDDs
- ❖ Some Conclusions on BDDs
- ❖ BDD Packages

Representing Circuit Functions

- ❖ **Functions**
  - All outputs of 4-bit adder
  - Functions of data inputs



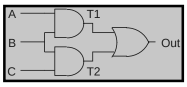
- ❖ **Shared Representation**
  - Graph with multiple roots
  - 31 nodes for 4-bit adder
  - 571 nodes for 64-bit adder
  - *Linear growth*



Generating OBDD from Network

Task: Represent output functions of gate network as OBDDs.

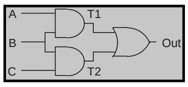
Network



Generating OBDD from Network

Task: Represent output functions of gate network as OBDDs.

Network



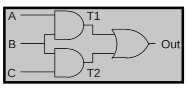
Evaluation

```
A   ←  new_var ("a");
B   ←  new_var ("b");
C   ←  new_var ("c");
T1  ←  And (A, B);
T2  ←  And (B, C);
Out ←  Or (T1, T2);
```

Generating OBDD from Network

Task: Represent output functions of gate network as OBDDs.

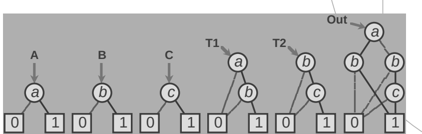
Network

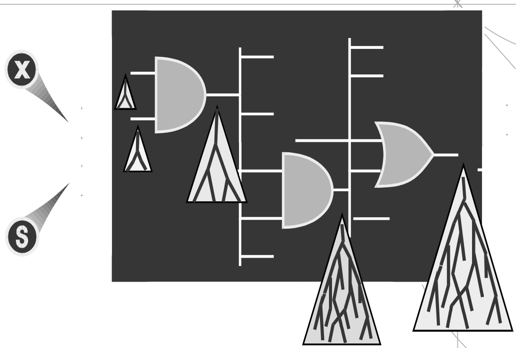


Evaluation

```
A   ←  new_var ("a");
B   ←  new_var ("b");
C   ←  new_var ("c");
T1  ←  And (A, B);
T2  ←  And (B, C);
Out ←  Or (T1, T2);
```

Resulting Graphs





- **Strategy**
  - **Represent data as set of OBDDs**
    - **Identical variable orderings**
  - **Express solution method as sequence of symbolic operations**
    - **Sequence of constructor & query operations**
    - **Similar style to on-line algorithm**
  - **Implement each operation by OBDD manipulation**
    - **Do all the work in the constructor operations**
- **Key Algorithmic Properties**
  - **Arguments are OBDDs with identical variable orderings**
  - **Result is OBDD with same ordering**
  - **Each step polynomial complexity**

Build BDDs: The Apply Procedure

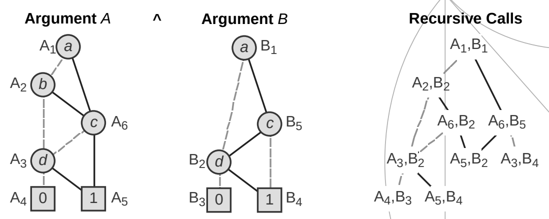
- **Given:**
  - **two BDDs one for f and one for g**
  - **the logical operator op**
- **To build**
  - **r = f op g**
    (and of two BDDs, or of two BDDs etc.) call:
- **Do the following:**
  - **Init computed table CT**
  - **r = APPLY (f, g)**

with:

APPLY (f, g)

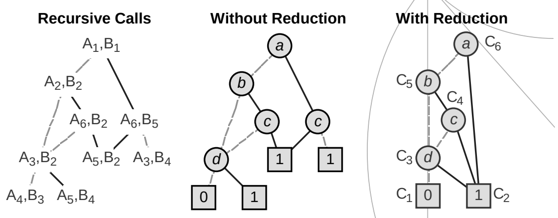
1. **IF CT(f, g) ≠ empty THEN return (CT (f, g))**
2. **ELSE if f and g ∈{ 0, 1} THEN r = op (f, g)**
3. **ELSE if topVar(f) = topVar(g) THEN**
   - **r = ITE (topVar (f), APPLY (T(f), T(g)), APPLY (E(f), E(g)))**
4. **ELSE if topVar(f) < topVar(g) THEN**
   - **r = ITE (topVar (f), APPLY (T(f), g), APPLY (E(f), g))**
5. **ELSE /* topVar(f) > topVar(g) */**
   - **r = ITE (topVar (g), APPLY (f, T(g)), APPLY (f, E(g)))**
6. **put r in G**
7. **return (r)**

Execution Example



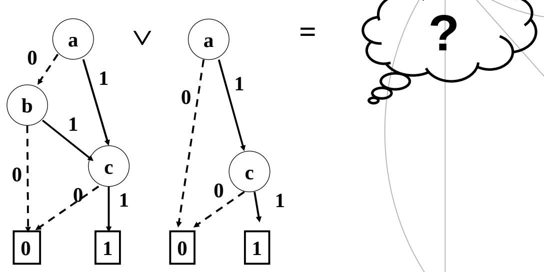
- **Optimizations**
  - **Dynamic programming**
  - **Early termination rules**

Result Generation

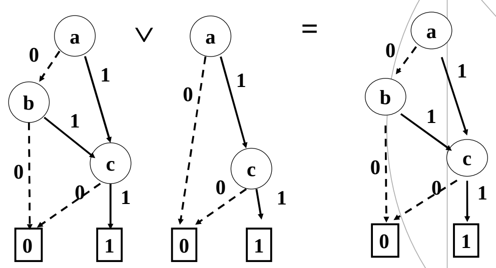


- **Recursive calling structure implicitly defines unreduced BDD**
- **Apply reduction rules bottom-up as return from recursive calls**
- **Do not create new result node if both brances equal (return that result) or if equivalent node already exists in reduce table. (The apply function is also memoized.)**

Example



Example



Outline

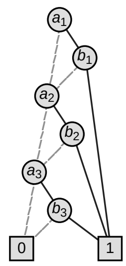
- ❖ Binary Decision Diagrams: Fundamentals
- ❖ Generation of BDDs from Network
- ❖ Variable Ordering Related Problems
- ❖ Complex Operations with BDDs
- ❖ Some Conclusions on BDDs
- ❖ BDD Packages

Effect of Variable Ordering

$F(a_1, a_2, a_3, b_1, b_2, b_3) = (a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$

Effect of Variable Ordering

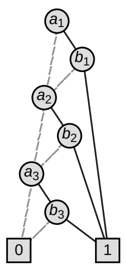
$F(a_1, a_2, a_3, b_1, b_2, b_3) = (a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$



Effect of Variable Ordering

$F(a_1, a_2, a_3, b_1, b_2, b_3) = (a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$

Good Ordering

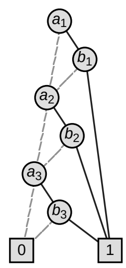


Linear Growth

Effect of Variable Ordering

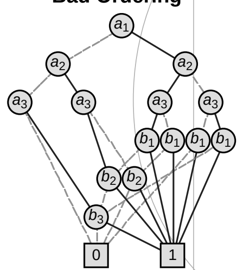
$F(a_1, a_2, a_3, b_1, b_2, b_3) = (a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$

Good Ordering



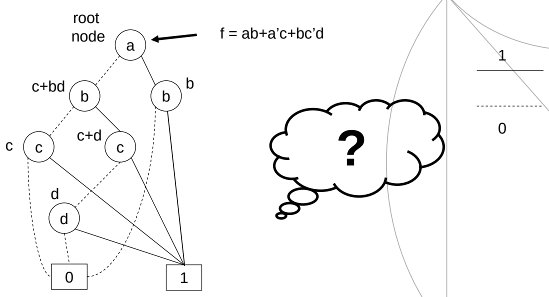
Linear Growth

Bad Ordering



Exponential Growth

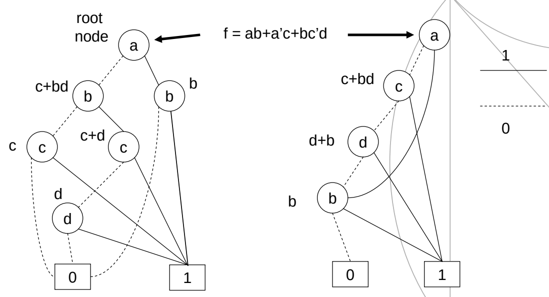
Exercise



Given the BDD with variable order a, b, c, d

Represents it with the order a, c, d, b.

Exercise



Given the BDD with variable order a, b, c, d

Represents it with the order a, c, d, b.

Sample Function Classes

Function Class	**Best**	**Worst**	**Ordering Sensitivity**
ALU (Add/Sub)	linear	exponential	High
Symmetric	linear	quadratic	None
Multiplication	exponential	exponential	Low

- ❖ **General Experience**
  - ◆ **Many tasks have reasonable OBDD representations**
  - ◆ **Algorithms remain practical for up to 5,000,000 node OBDDs**
  - ◆ **Heuristic ordering methods generally satisfactory**

Consideration on Variable Ordering

- ❖ **Variable order is fixed**
  For each path from root to terminal node the order of "input" variables is exactly the same
- ❖ **Strong dependency of the BDD size (terms of nodes) and variable ordering**
- ❖ **Ordering algorithm:**
  - ◆ **Co-NP complete problem - heuristic approaches**
  - ◆ **Static Variable Ordering Heuristic**
  - ◆ **Dynamic Variable Ordering Heuristic**
  - ◆ **ROBDDs - Reduced Ordered Binary DDs (BDDs!)**

Static Variable Ordering

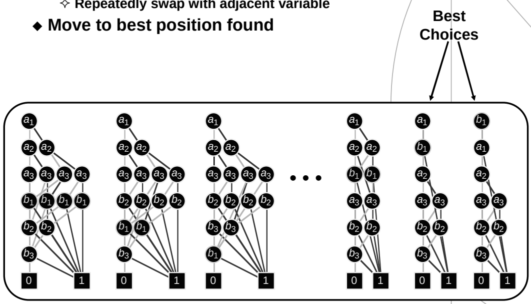
- ❖ **Different heuristic introduced over the years**
- ❖ **Usually based on the circuit structure**
  - ◆ **E.g., depth-first visit from the outputs**
- ❖ **Sufficient for "static problems"**
- ❖ **Insufficient for "dynamic requirements"**

Dynamic Variable Reordering

- ❖ **First Introduced by Richard Rudell, Synopsys, 1991**
- ❖ **Periodically Attempt to Improve Ordering for All BDDs**
  - ◆ **Part of garbage collection**
  - ◆ **Move each variable through ordering to find its best location**
- ❖ **Has Proved Very Successful**
  - ◆ **Time consuming but effective**
  - ◆ **Especially for sequential circuit analysis**

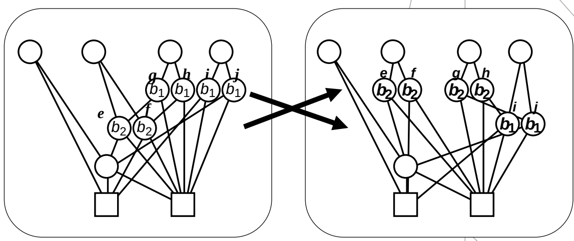
Dynamic Reordering By Sifting

- Choose candidate variable
- Try all positions in variable ordering
  - Repeatedly swap with adjacent variable
- Move to best position found



Swapping Adjacent Variables

- Localized Effect
  - Add / delete / alter only nodes labeled by swapping variables
  - Do not change any incoming pointers

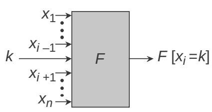


Outline

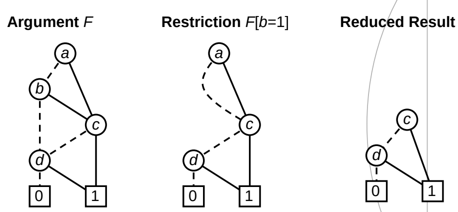
- Binary Decision Diagrams: Fundamentals
- Generation of BDDs from Network
- Variable Ordering Related Problems
- Complex Operations with BDDs
- Some Conclusions on BDDs
- BDD Packages

Restriction

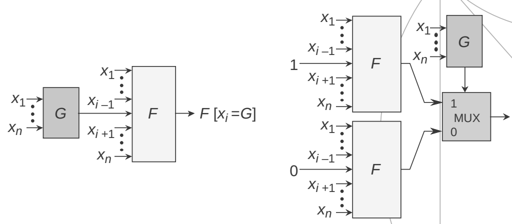
- Concept
  - Effect of setting function argument x_i to constant k (0 or 1).
  - Also called Cofactor operation (UCB)
    - F_x equivalent to F [x=1]
    - $F_{\neg x}$ equivalent to F [x=0]



Restriction Execution Example



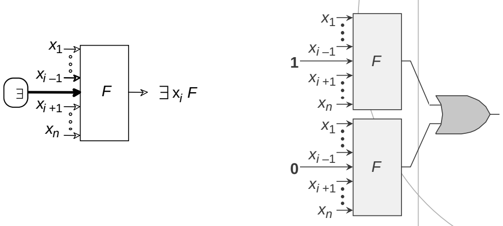
Functional Composition



- Create new function by composing functions F and G.
- Useful for composing hierarchical modules.

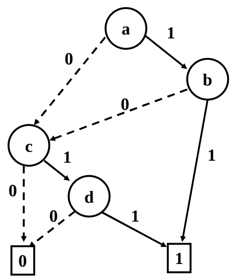
Existential Variable Quantification

- $\exists_b$ **f = f |** $_{b=0}$ $\vee$ **f |** $_{b=1}$
  - **Eliminate dependency on some argument**
  - **Efficient algorithm for quantifying over a set of variables**



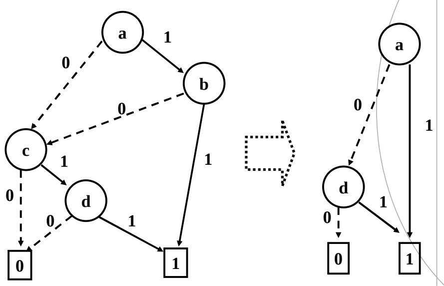
Example

$\exists_{(b,c)}. ((a \wedge b) \vee (c \wedge d)) = ?$



Example

$\exists_{(b,c)}. ((a \wedge b) \vee (c \wedge d)) = a \vee d$



Universal Variable Quantification

- $\forall_b$ **f = f |** $_{b=0}$ $\wedge$ **f |** $_{b=1}$
  - **Obtained with existential quantification combine with AND**

Outline

- **Binary Decision Diagrams: Fundamentals**
- **Generation of BDDs from Network**
- **Variable Ordering Related Problems**
- **Complex Operations with BDDs**
- **Some Conclusions on BDDs**
- **BDD Packages**

What's good about BDDs?

- **Powerful Operations**
  - **Creating, manipulating, testing**
  - **Each step polynomial complexity**
    - **Graceful degradation**
- **Generally Stay Small Enough**
  - **Especially for digital circuit applications**
  - **Given good choice of variable ordering**
- **Extremely useful in practice**
- **(Till 5 years ago) Weak Competition**
  - **No other method comes close in overall strength**
  - **Especially with quantification operations**

What's bad about BDDs?

- **Some formulas do not have small representation! (e.g., multipliers)**
- **BDD representation of a function can vary exponentially in size depending on variable ordering; users may need to play with variable orderings (less automatic)**
- **Size limitations: a big problem**
- **(Last 5 years) Competitive Approach: CNF representation + SATisfiability solvers**

Thoughts on Algorithms Research

- **Need to be Willing to Attack Intractable Problems**
  - **Many real-world problems NP-hard**
  - **No approximations for verification**
- **Who Works on These?**
  - **Mostly people in application domain**
    - **Most work on BDDs in computer-aided design conferences**
  - **Not by people with greatest talent in algorithms**
    - **Probably many ways they could improve things**
  - **Fundamental dilemma**
    - **Can only make weak formal statements about efficiency**
    - **Utility demonstrated empirically**

Outline

- **Binary Decision Diagrams: Fundamentals**
- **Generation of BDDs from Network**
- **Variable Ordering Related Problems**
- **Complex Operations with BDDs**
- **Some Conclusions on BDDs**
- **BDD Packages**

A few BDD Packages

- **Brace, Rudell, Bryant:** KBDD
  - Carnegie Mellon, 1990
  - Synopsys, 1993 on
  - Digital, Compaq, Intel, 1993 on
- **Long:** KBDD
  - Carnegie Mellon, 1993
  - AT&T, 1995 on
- **Armin Biere**: ABCD
  - Carnegie Mellon / Universität Karlsruhe
- **Olivier Coudert**: TiGeR
  - Synopsys / Monterey Design Systems
- **Geert Janssen**: EHV
  - Eindhoven University of Technology

- **Geert Janssen**: EHV
  - Eindhoven University of Technology
- **Rajeev K. Ranjan**: CAL
  - UCB, Synopsys
- **Bwolen Yang**: PBF
  - Carnegie Mellon
- **Stefan Horeth**: TUDD
  - University TU Darmstadt
  - **http://marple.rs.e-technik.tu-darmastadt.de/~sth**
- **Fabio Somenzi**: CUDD
  - University of Colorado
  - **http://vlsi.colorado.edu/~fabio**