Boolean Algebra and Binary Decision Diagrams

Profs. Sanjit Seshia & Kurt Keutzer
EECS
UC Berkeley

With thanks to Rob Rutenbar, CMU

1

Today's Lecture

- Boolean algebra basics
- Binary Decision Diagrams
  - Representation, size
  - Building BDDs
- Finish up with equivalence checking

S. Seshia

2

Recap

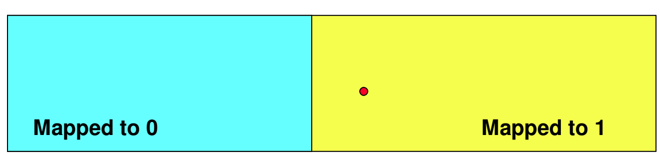
What is a

- Literal?
- Cube?
- Minterm?

Boolean function

A Boolean function F of n variables $x_1, x_2, \ldots, x_n$

$$F : \{0,1\}^n \rightarrow \{0,1\}$$



Cofactors

A Boolean function F of n variables $x_1, x_2, \ldots, x_n$

$$F : \{0,1\}^n \rightarrow \{0,1\}$$

Suppose we define new Boolean functions of n-1 variables as follows:

$$F_{x_1}(x_2, \ldots, x_n) = F(1, x_2, x_3, \ldots, x_n)$$

$$F_{x_1'}(x_2, \ldots, x_n) = F(0, x_2, x_3, \ldots, x_n)$$

F_{x_1} and $F_{x_1'}$ are cofactors of F.

What does their input state space look like?

Examples of Cofactors

$F(x, y, z) = xy + xz' + y(x'z + z')$

What's F_x ? $y + z' + yz'$

$F_{x'}$? $yz + yz'$

OK, so why are cofactors useful?

Analogy: Taylor series expansion

Represent complex function using simpler functions

$f(x) = f(0) + x\, f'(0) + x^2/2!\, f''(0) + \ldots$

Anything like this for Boolean functions?

ANS: Yes, using cofactors!

Shannon Expansion

$$F(x_1, \ldots, x_n) = x_i \,.\, F_{x_i} + x_i' \,.\, F_{x_i'}$$

Proof?

Shannon expansion with many variables

$F(x, y, z, w) = xy\, F_{xy} + x'y\, F_{x'y} + xy'\, F_{xy'} + x'y'\, F_{x'y'}$

Assuming previous slide, how would you derive the above?

Is Cofactoring commutative? i.e. $(F_x)_y = (F_y)_x$?

S. Seshia

9

Properties of Cofactors

- Suppose you construct a new function H from two existing functions F and G: e.g.,
  - H = F'
  - H = F.G
  - H = F + G
  - Etc.
- What is the relation between cofactors of H and those of F and G?

S. Seshia

10

Very Useful Property

- Cofactor of NOT is NOT of cofactors
- Cofactor of AND is AND of cofactors
- …

- Works for any binary operator

Back to BDDs: Recap

Truth Table

↓

Binary Decision Tree

↓

Binary Decision Diagram (BDD)

↓

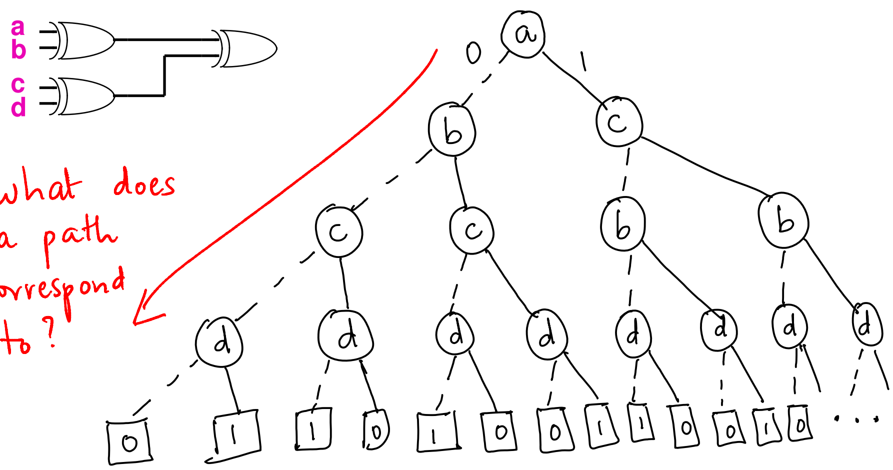
Ordered Binary Decision Diagram (OBDD)

↓

Reduced Ordered Binary Decision Diagram (ROBDD, simply called BDD)

What steps happen at each arrow?

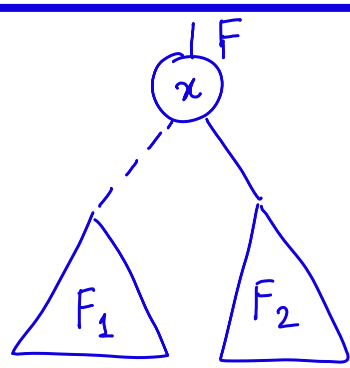
Example: Odd Parity Function



what does a path correspond to?

Binary Decision Tree

Nodes & Edges

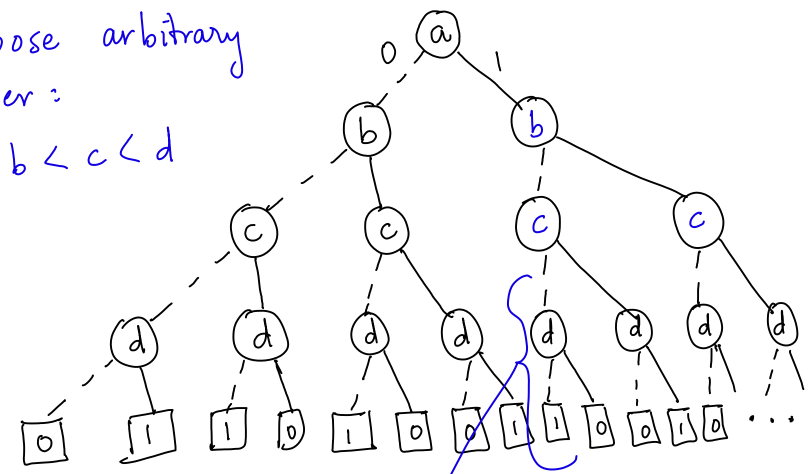


- How is F related to x, F_1, F_2?

Ordering

Impose arbitrary order:

$a < b < c < d$



Why didn't this part change?

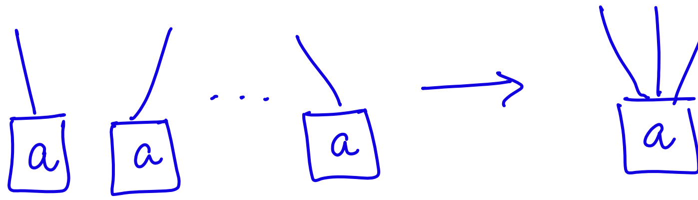
Reduction

Identify Redundancies

3 Rules:

1. **Merge equivalent leaves**
2. **Merge isomorphic nodes**
3. **Eliminate redundant tests**

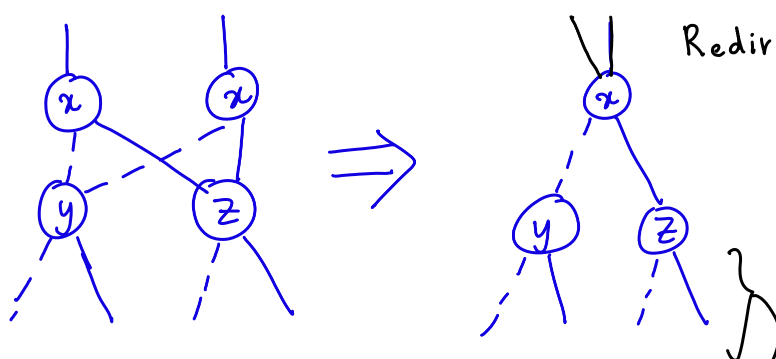
Merge Equivalent Leaves



"a" is either 0 or 1

S. Seshia 17

Merge Isomorphic Nodes

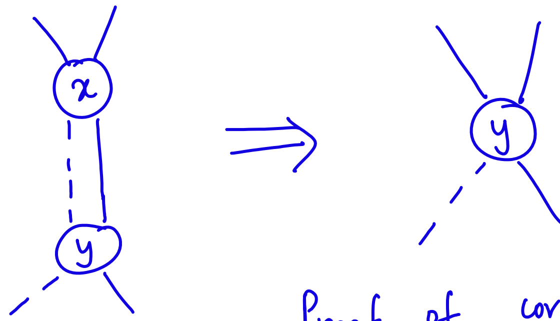


Redirect

stays same down here

S. Seshia 18

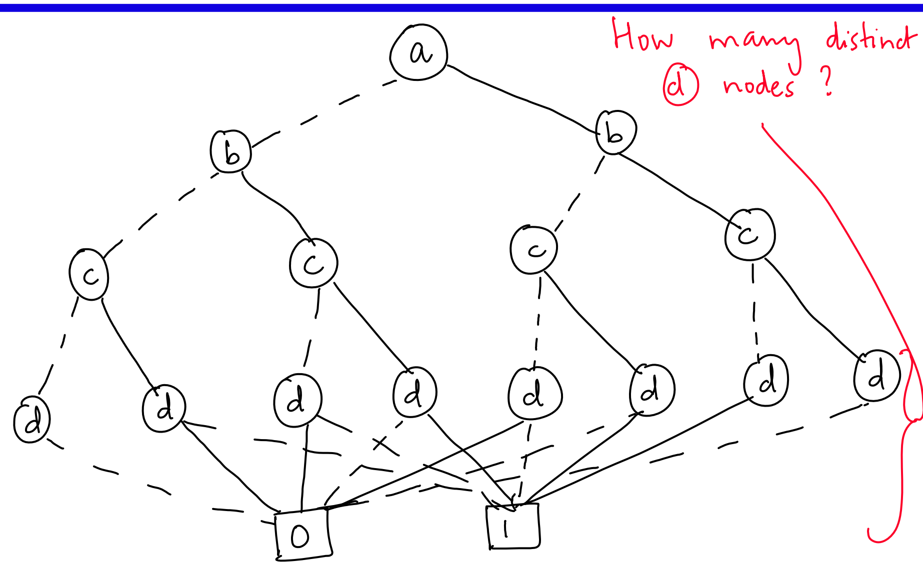
Eliminate Redundant Tests



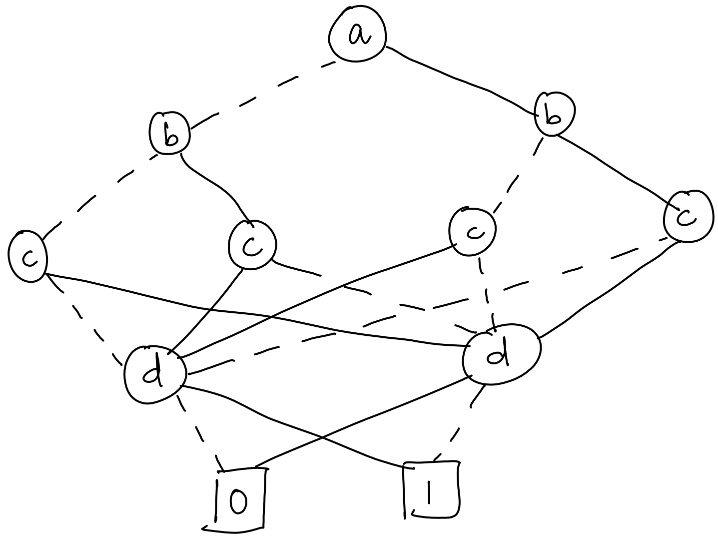
Proof of correctness ?

Example

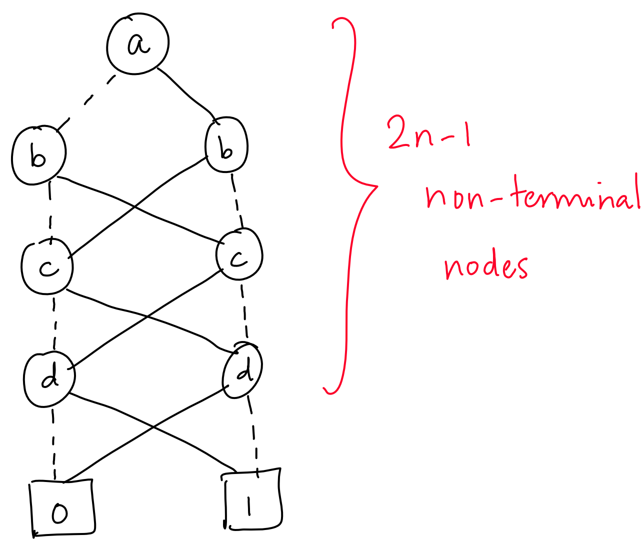
How many distinct d nodes ?



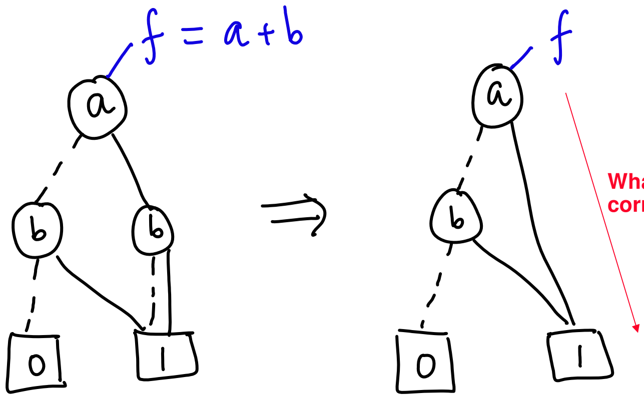
Example



Final ROBDD for Odd Parity Function

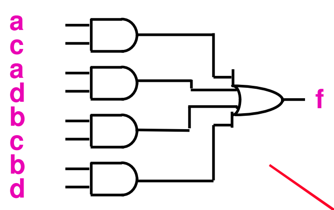


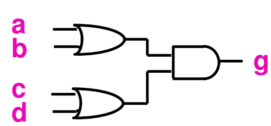
Example of Rule 3

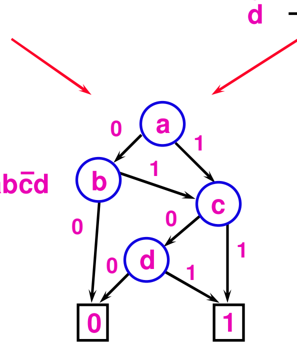


What does a path correspond to?

ROBDDs are Canonical







$f = ac + \bar{a}bc + a\bar{c}d + \bar{a}b\bar{c}d$

disjoint cover

ordering
a b c d

Proof that ROBDDs are canonical

Theorem (R. Bryant): If G, G' are ROBDD's of a Boolean function f with k inputs, using same variable ordering, then G and G' are identical.

S. Seshia

25

ROBDDs are Canonical - use 1

Given an ordering, a logic function has a unique ROBDD.

Given two circuits, *checking their equivalence* reduces to a Directed Acyclic Graph isomorphism check between their respective ROBDDs

- can be done in linear time in $|G_1|$ (= $|G_2|$).
- How big can a ROBDD get?

S. Seshia

26

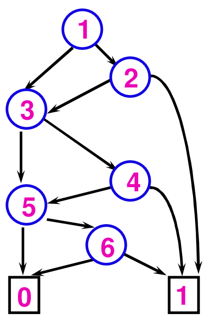
Sensitivity to Ordering

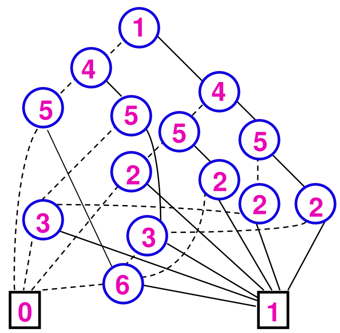
Given a function with n inputs, one input ordering may require exponential # vertices in ROBDD, while other may be linear in size.

$f = x_1 x_2 + x_3 x_4 + x_5 x_6$

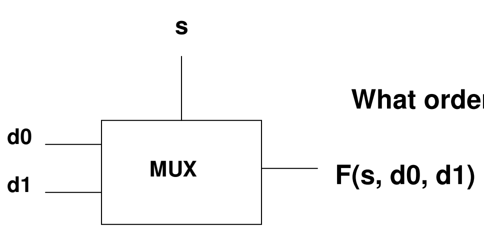
$x_1 < x_2 < x_3 < x_4 < x_5 < x_6$

$x_1 < x_4 < x_5 < x_2 < x_3 < x_6$





Another Ordering Example

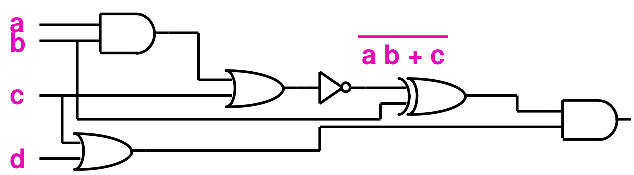


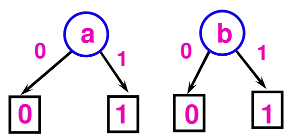
What ordering to pick for F and why?

ROBDD Construction

Given ordering and multilevel network.

ROBDD of a b





Begin with ROBDDS for primary inputs

Proceed through network, constructing the ROBDD for each gate output, by applying the gate operator to the ROBDDs of the gate inputs

Applying an Operator to BDDs

Two options:

1. **Construct an operator for each logic operator: AND, OR, NOT, EXOR, …**
2. **Build a few core operators and define everything else in terms of those**

Advantage of 2:
- Less programming work
- Easier to add new operators later by writing “wrappers”

Core Operators

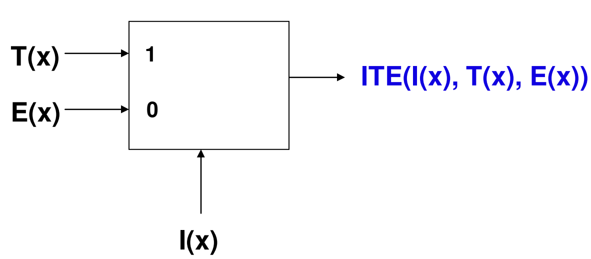
Just two of them!

1. Restrict(Function F, variable v, constant k)
   - Shannon cofactor of F w.r.t. v=k
2. ITE(Function I, Function T, Function E)
   - "if-then-else" operator

S. Seshia 31

ITE

- Just like:
  - "if then else" in a programming language
  - A mux in hardware
- ITE(I(x), T(x), E(x))
  - If I(x) then T(x) else E(x)



S. Seshia 32

The ITE Function

ITE(I(x), T(x), E(x))

=

I(x) . T(x) + I'(x). E(x)

What good is the ITE?

How do we express

- NOT?
- OR?
- AND?

How do we implement ITE?

Divide and conquer!

Use Shannon cofactoring…

- Recall: Operator of cofactors is Cofactor of operators…

ITE Algorithm

```
ITE (bdd I, bdd T, bdd E) {
  if (terminal case) { return computed result; }
  else { // general case
     Let x be the topmost variable of I, T, E;
     PosFactor = ITE(I_x , T_x , E_x) ;
     NegFactor = ITE(I_x' , T_x' , E_x');
     R = new node labeled by x;
     R.low = NegFactor;
     R.high = PosFactor;
     Reduce(R);
     return R;
}
```

Terminal Cases

- ITE(1, T, E) =
- ITE(0, T, E) =
- ITE(I, T, T) =
- ITE(I, 1, 0) =
- …

General Case

- Still need to do cofactor (Restrict)
- How hard is that?
  - Which variable are we cofactoring out? (2 cases)

ITE Algorithm – Complexity?

```
ITE (bdd I, bdd T, bdd E) {
  if (terminal case) { return computed result; }
  else { // general case
     Let x be the topmost variable of I, T, E;
     PosFactor = ITE(I_x, T_x, E_x) ;
     NegFactor = ITE(I_x', T_x', E_x');
     R = new node labeled by x;
     R.low = NegFactor;
     R.high = PosFactor;
     Reduce(R);
     return R;
}
```

How many ITE calls can we make?

Practical Issues

- Previous calls to ITE are cached
  - "memoization"
- Every BDD node created goes into a "unique table"
  - Before creating a new node R, look up this table
  - Avoids need for reduction

ROBDD-based equivalence checking

Given circuits C1 and C2 to be verified for equivalence

A1) create the ``comparison circuit'' D1

A2) find a variable ordering for the ROBDD for D1

A3) build the ROBDD and check for 0

or

B1) find a variable ordering for the ROBDD's of C1, C2

B2) build the ROBDD for each of C1, C2

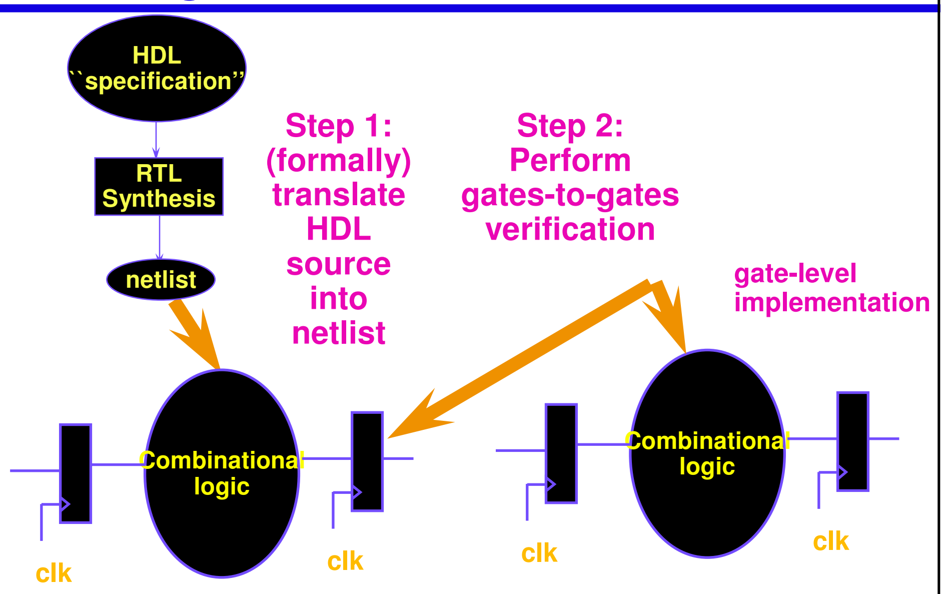
B3) Check to see that the DAGs are isomorphic

Putting it all together

Current formula requires:

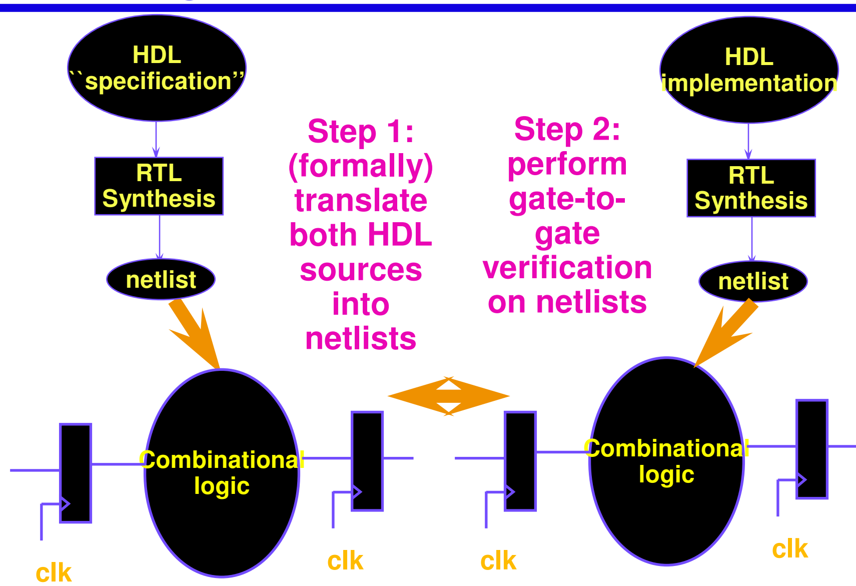
- Ability to associate FF's from the two circuits
- Exploiting structural similarity/check-points
- Applying whatever works:
  - Test techniques, SAT for more regular structures
  - BDD for more random
  - Mix and match

Solving RTL-to-Gates Verification



S. Seshia 43

Solving RTL-to-RTL Verification



S. Seshia 44

Current status of equivalence checking

Equivalence checking is one of the great successes of EDA in the late 90's

Equivalence checkers are now able to routinely verify complex (>10M gate) integrated circuit designs

Coupled with static timing analysis it has enabled "static-signoff"

Current technology leaders are Cadence Verplex and Synopsys Formality. Good proprietary (e.g. IBM/verity) solutions exist

Successful equivalence checkers must orchestrate a number of different approaches

- syntactic equivalence
- automatic test pattern generation-like approaches
- BDD-based techniques

A few open problems remain:

- retimed circuits
- circuits with differing state assignments