Hints on Specific Techniques

P.Camurati G.Cabodi S.Nocco S.Quer

Formal Methods Group
Department of Computer Engineering
Politecnico di Torino
Torino, Italy

Outline

- Symbolic Function Representation
- Combinational BDD-based verification
- Sequential BDD-based verification
- Satisfiability (SAT)
- Combinational Satisfiability-based verification
- Sequeential Satisfiability-based verification

Outline

- Symbolic Function Representation
- Combinational BDD-based verification
- Sequential BDD-based verification
- Satisfiability (SAT)
- Combinational Satisfiability-based verification
- Sequeential Satisfiability-based verification

Representation of Boolean Functions

- What do we need?
  - A good data structure for Boolean formulas !!!
- Why?
  - To represent the problem
  - To to manipulate the representation used, i.e., to perform Boolean Reasoning (e.g., a decision procedure to decide about SAT or UNSAT)
- Representation Methods
  - Classical Methods
    - Canonical Forms
    - NON Canonical Forms
  - Non-Classical Methods

Classical Canonical Methods

- Truth Table
  - F = Graphical/Tabular Representation
- Canonical Disjunctive Normal Form (cDNF)
  - F = (x1* ∧ x2* ∧ ... ∧ xn*) ∨ ... ∨ (x1* ∧ x2* ∧ ... ∧ xn*)
- Canonical Conjunctive Normal Form (cCNF)
  - F = (x1* ∨ x2* ∨ ... ∨ xn*) ∧ ... ∧ (x1* ∨ x2* ∨ ... ∨ xn*)

Example

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

- Truth Table
- DNF
  - F = (¬x1 ∧ x2 ∧ x3) ∨ (x1 ∧ ¬ x2 ∧ x3) ∨ (x1 ∧ x2 ∧ x3)
- CNF
  - F = (x1 ∨ x2 ∨ x3) ∧(x1 ∨ x2 ∨ ¬ x3) ∧(x1 ∨ ¬ x2 ∨ x3) ...

- **Pros**
  - Unique representation (one and only for each function)
  - Constant Time Comparison (same representation)
- **Cons**
  - Exponential Size
  - Complex Resolution Algorithms
  - Satisfiability is NP-complete (Cook) (i.e., resolution algorithms require exponential time)
  - Examples
    - DNF → satisfiability requires polynomial time, tautology is co-NP complete
    - CNF → … vice-versa …
    - Conversion CNF ↔ DNF is exponential

Classical Non Canonical Methods

- **Disjunctive Normal Form (DNF)**
  - F = (x1* ∧ ... <some i missing> ... ∧ xn*) ∨ ... ∨ (x1* ∧ ... ∧ xn*)
- **Conjunctive Normal Form (CNF)**
  - F = (x1* ∨ ... <some i missing> ... ∨ xn*) ∧ ... ∧ (x1* ∨ ... ∨ xn*)

- **Pros**
  - Non-Exponential Representation's Size
- **Cons**
  - Non-Unique representation (more representations for each function)
  - Complex Algorithms for Comparison
  - Complex Algorithms for Conversions

Non Classical Representation

- **Decision Diagrams**
  - BDDs - Binary Decision Diagrams
  - ZBDDs - Zero Suppressed Binary Decision Diagrams
  - Etc.
- **Boolean Circuits**
  - AIGs – And Inverter Graphs
  - RBCs – Reduced Boolean Circuits
  - Etc.

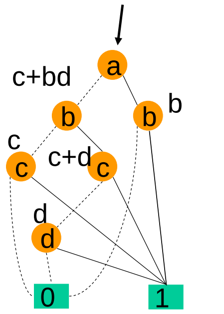
Binary Decision Diagrams

- ***Idea from 70s (maybe earlier)***
- ***Adapted by Bryant '86***
- ***Take a formula***
- ***Make decision tree for fixed variable order***
- ***Reduction rules***
  - *Merge duplicate nodes*
  - *Both children point to same node - remove redundant node*

Binary Decision Diagrams (BDD)

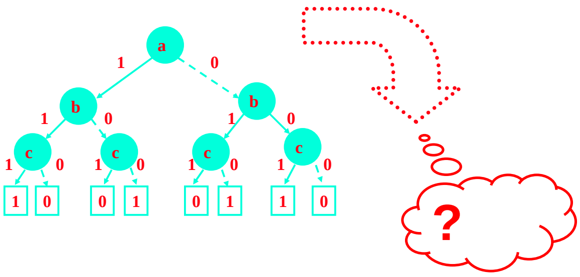
- ***Graph representation of a Boolean function f***
  - vertices represent decision nodes for variables
  - two children represent the two subfunctions
  - f(x = 0) and f(x = 1) (cofactors)
  - can make a BDD representation canonical

f = ab+a'c+a'bd



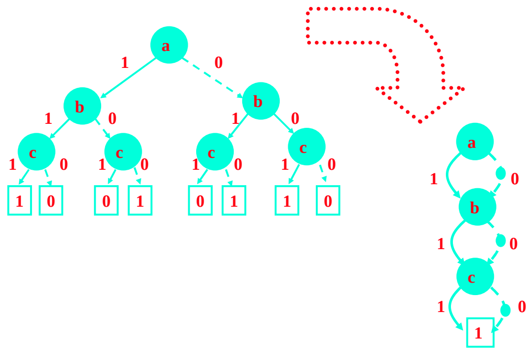
Example 1

- F (a,b,c) = (a ⊕ b) ⊕ c



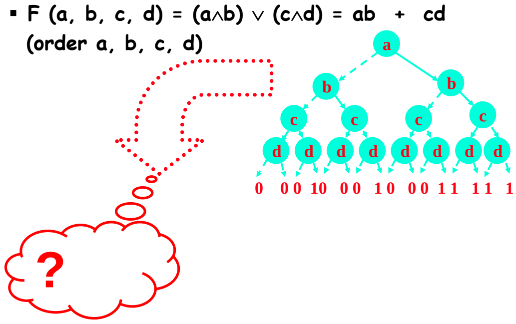
Example 1

- F (a,b,c) = (a ⊕ b) ⊕ c



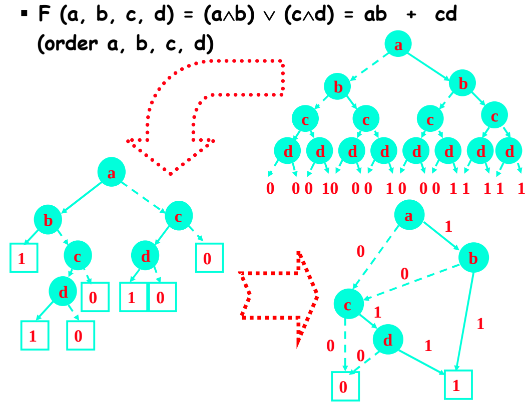
Example 2

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



Example 2

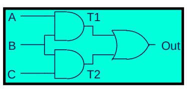
- F (a, b, c, d) = (a∧b) ∨ (c∧d) = ab + cd
  (order a, b, c, d)



Generating BDD from Network

- Task: Represent output functions of gate network as BDDs

Network

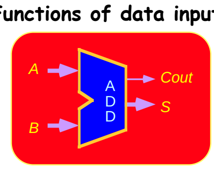


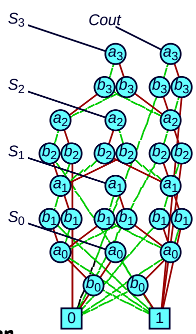
Evaluation

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

Representing Circuit Functions

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





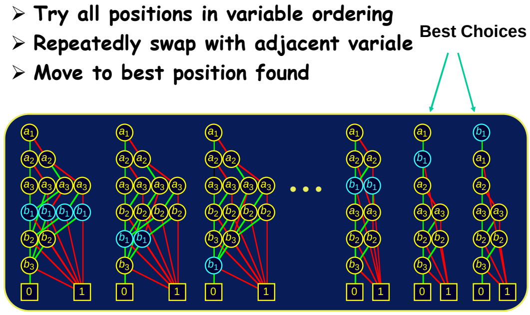
- Shared Representation
  - Graph with multiple roots
  - 31 nodes for 4-bit adder
  - 571 nodes for 64-bit adder
  - Linear growth

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!)

Dynamic Reordering By Sifting

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



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 10 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

Outline

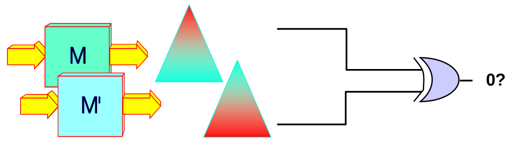
- Symbolic Function Representation
- Combinational BDD-based verification
- Sequential BDD-based verification
- Satisfiability (SAT)
- Combinational Satisfiability-based verification
- Sequeential Satisfiability-based verification

Combinational EC (1/2)

- Industrial EC checkers often use an combinational EC paradigm
  - Sequential EC is too complex, can only be applied to design with a few hundred state bits
  - Combinational methods scale linearly with the design size for a given fixed size and "functional complexity" of the individual cones

Combinational EC (2/2)

- Still, pure BDDs as plain SAT solver cannot handle all cones
  - BDDs can be built for about 80% of the cones of high-speed designs
  - less for complex ASICs
  - plain SAT blows up on a "Miter" structure
- Contemporary method highly exploit structural similarity of designs to be compared

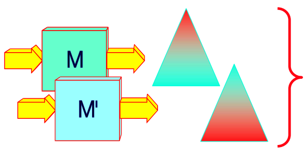


- Verification (base method)
  - Compare output's BDDs

$M_i == M'_i \ , \forall i$?

⇩

$(M_i \oplus M'_i) == 0 \ , \forall i$?

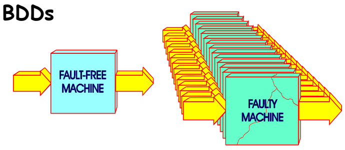


$M_i == M'_i \ , \forall i$?

⇩

$(M_i \oplus M'_i) == 0 \ , \forall i$?

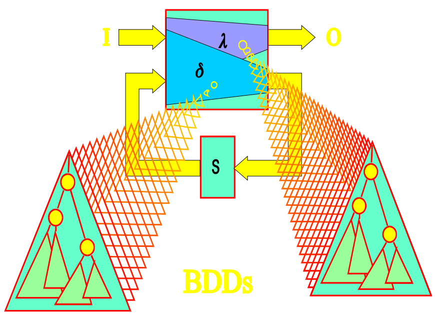
- Verification (base method)
  - Compare output's BDDs
- Testing
  - Fault
  - Stuck-at 0/1



Outline

- Symbolic Function Representation
- Combinational BDD-based verification
- Sequential BDD-based verification
- Satisfiability (SAT)
- Combinational Satisfiability-based verification
- Sequeential Satisfiability-based verification

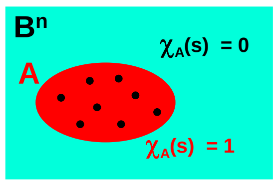
Function Representation

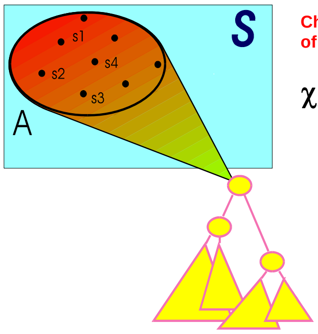


(State) Set Representation

- Given a set *A*
- We define the Characteristic Function $\chi_A(s)$ of the set *A* as

$$\chi_A(s) = \begin{cases} 1 & \text{IFF } s \in A \\ 0 & \text{IFF } s \notin A \end{cases}$$





Characteristic Function of set A

$\chi_A(s)$ = 1 IFF $s \in A$
= 0 IFF $s \notin A$

Image and inverse image

$Img(f, X) = f(X) = \{ y \in B^m \mid x \in X \wedge y = f(x) \}$

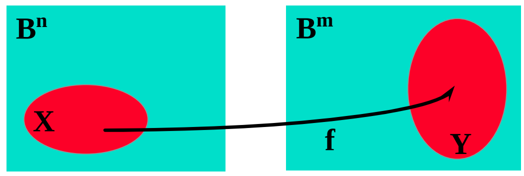


Image and inverse image

$Img(f, X) = f(X) = \{ y \in B^m \mid x \in X \wedge y = f(x) \}$

$PreImg(f, Y) = f^{-1}(Y) = \{ x \in B^n \mid y \in Y \wedge y = f(x) \}$

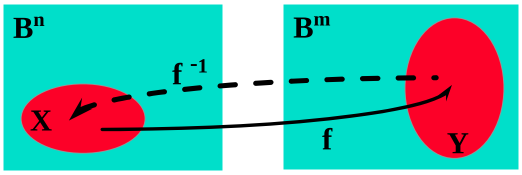
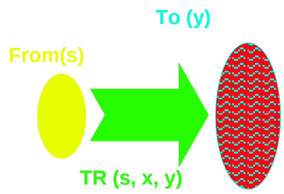
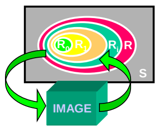


Image Computation

To (y) = Img (TR, From) = $\exists_{s,x}$ [TR (s, x, y) · From(s)]

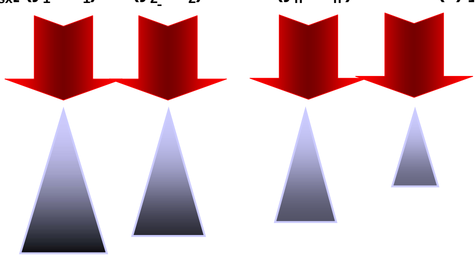


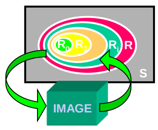


To (y) =

= $\exists_{sx}$[TR (s,x,y) · From (s)]

= $\exists_{sx}$[$(y_1 \equiv \delta_1)$ · $(y_2 \equiv \delta_2)$ · … · $(y_n \equiv \delta_n)$ · From (s)]



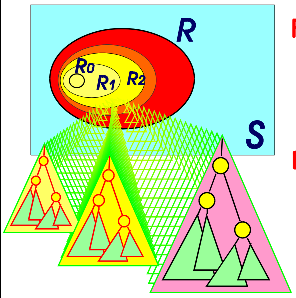


To (y) =

= $\exists_{sx}$[TR (s,x,y) · From (s)]

= $\exists_{sx}$[$(y_1 \equiv \delta_1)$ · $(y_2 \equiv \delta_2)$ · … · $(y_n \equiv \delta_n)$ · From (s)]

State Traversal



Forward Traversal

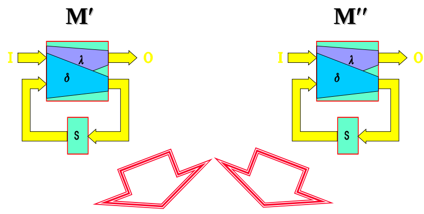
R_0 = *Initial State Set*

R_{i+1} = R_i + Img (TR, R_i)

Backward Traversal

R_0 = *Initial State Set*

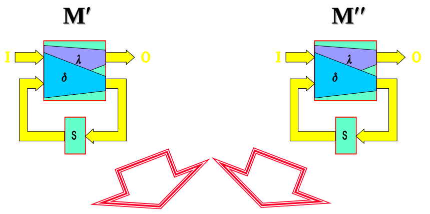
R_{i+1} = R_i + PreImg (TR, R_i)



Product Machine: M = M′ x M″

S_0 ⇐ Mutually Reachable ⇒ ($\lambda' \neq \lambda''$)

for equivalence checking

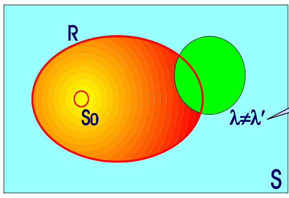


Product Machine: M = M′ x M″

S_0 ⇐ Mutually Reachable ⇒ bad set f state

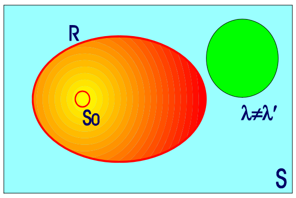
for (invariant) model checking

Exact Forward Traversal



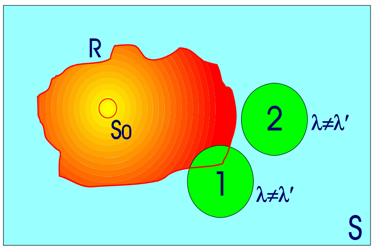
M ≠ M'

Bad set of state (model checking)



M ≡ M'

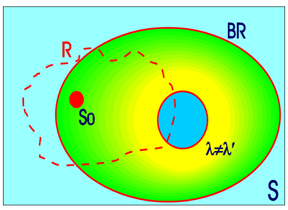
Problems



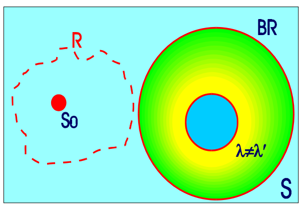
R is

- too large
- too difficult to evaluate

Exact Backward Traversal

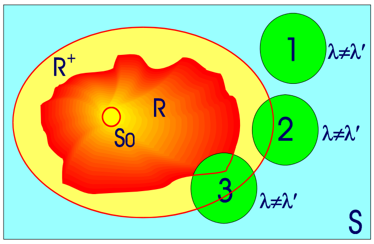


M ≠ M'



M ≡ M'

Aproximate Reachability



R^{+} = over-estimation of R

Verification

1. Equivalent in R & R^{+}
2. NOT Equivalent in R^{+} Equivalent in R
3. NOT Equivalent in R & R^{+}

Outline

- Symbolic Function Representation
- Combinational BDD-based verification
- Sequential BDD-based verification
- Satisfiability (SAT)
- Combinational Satisfiability-based verification
- Sequential Satisfiability-based verification

Boolean Satisfiability (SAT)

- Given a suitable representation for a Boolean function *f* (*X*)
  - Find an assignment *X** such that *f* (*X**) = 1

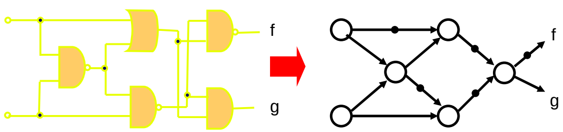
  OR
  - *Prove that such an assignment does not exist,* i.e., *f*(*X*) = 0 for all possible assignments
- In the "classical" SAT problem, *f* (*X*) is represented as
  - Product-of-sums (POS)

  OR
  - Conjunctive normal form (CNF)

- SAT belongs to NP
  - There is a non-deterministic Touring Machine deciding SAT in polinomial time
  - On a real – deterministic computer this would require exponential time
- Many decision (yes/no) problems can be formulated either directly or indirectly in terms of Boolean Satisfiability

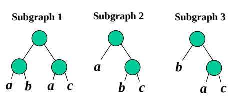
And Inverter Graphs (AIGs)

- Base data structure uses two-input AND function for vertices and INVERTER attributes at the edges (individual bit)
  - Use De'Morgan's law to convert OR operation etc.
- Hash table to identify and reuse structurally isomorphic circuits

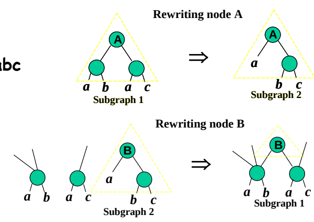


AIG Optimizations

- *AIG rewriting minimizes the number of AIG nodes without increasing the number of AIG levels*
- Pre-computing AIG subgraphs
  - Consider function f = abc



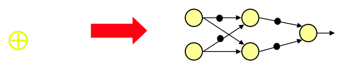
Rewriting AIG subgraphs



In both cases 1 node is saved

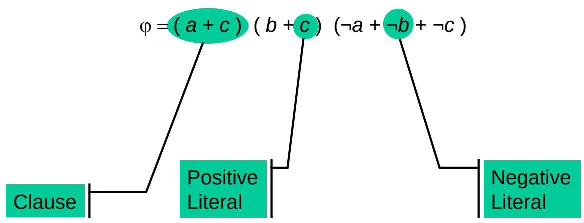
Circuit (AIG) to CNF

- **Naive conversion of circuit to CNF**
  - Multiply out expressions of circuit until two level structure
  - Example
    - $y = x_1 \oplus x_2 \oplus x_2 \oplus \ldots \oplus x_n$ (Parity function)
    - Circuit size is linear in the number of variables

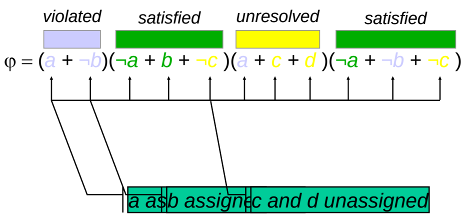


    - Generated chess-board Karnaugh map
    - CNF (or DNF) formula has 2^{n-1} terms (exponential in the # vars)
- **Better approach**
  - Introduce one variable per circuit vertex
  - Formulate the circuit as a conjunction of constraints imposed on the vertex values by the gates
  - Uses more variables but size of formula is linear in the size of the circuit

Conjunctive Normal Form (CNF)



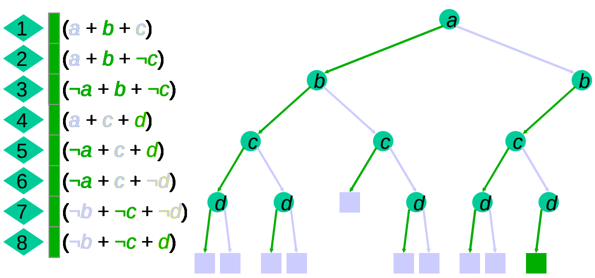
Literal & Clause Classification



Davis-Putnam (DP) Procedure

- Search for consistent assignment to entire cone of requested vertex by systematically trying all combinations (may be partial!!!)
- Keep a queue of vertices that remain to be justified
  - Pick decision vertex from the queue and case split on possible assignments
  - For each case
    - Propagate as many implications as possible
      - generate more vertices to be justified
      - if conflicting assignment encountered undo all implications and backtrack
    - Recur to next vertex from queue

Basic Case Splitting (Backtrack Search)



A SAT Example: Optimization of if-then-else chain

Original C code

```
if (!a && !b) h();
else if (!a) g();
else f();
```

```
if (!a) {
  if (!b) h();
  else g();
} else f();
```

Optimized C code

```
if (a) f();
else {
  if (!b) h();
  else g();}
```

```
if (a) f();
else if (b) g();
else h();
```

How to check if these are equivalent?

- **Represent procedures as independent Boolean variables**

  Original = if (¬a ∧ ¬ b) h();
  else if (¬a) g();
  else f();

  Optimized = if (a) f();
  else {
  if (¬b) h();
  else g();}
- **Compile the into Boolean formulae**

  if x then y else z = ITE (x,y,z) =(x ∧ y) (¬x ∧ z)
- **Check equivalence of Boolean formulae**

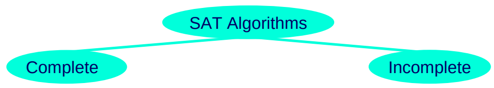
  Compile (Original) ≡ Compile (Optimized)

```
Original = if ¬a∧¬b then h else if ¬a then g else f
  = (¬a ∧ ¬b) ∧ h ∨ ¬(¬a ∧ ¬b) ∧
     if ¬a then g else h
  = (¬a ∧ ¬b) ∧ h ∨ ¬(¬a ∧ ¬b) ∧ (¬a ∧ g ∨ a ∧ f)
Optimized = if a then f else if b then g else h
  = (a ∧ f) ∨ ¬a ∧ if b then g else h
  = a ∧ f ∨ ¬a ∧ (b ∧ g ∨ ¬b ∧ h)

(¬a ∧ ¬b) ∧ h ∨ ¬(¬a ∧ ¬b) ∧ (¬a ∧ g ∨ a ∧ f)
≠
a ∧ f ∨ ¬a ∧ (b ∧ g ∨ ¬b ∧ h)
```

is satisfiable?

A Taxonomy of SAT Algorithms

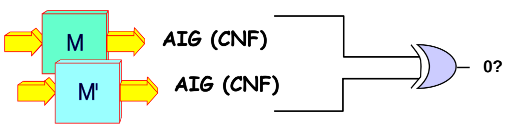


Complete	Incomplete
Can prove unsatisfiability	Cannot prove unsatisfiability
Backtrack search (DP)	Local search (hill climbing)
Resolution (original DP)	Continuous formulations
Stalmarck's method (SM)	Genetic algorithms
Recursive learning (RL)	Simulated annealing
BDDs	Tabu search
...	...

Outline

- Symbolic Function Representation
- Combinational BDD-based verification
- Sequential BDD-based verification
- Satisfiability (SAT)
- Combinational Satisfiability-based verification
- Sequential Satisfiability-based verification

Combinational EC



- **Verification (base method)**
  - Use a SAT solver

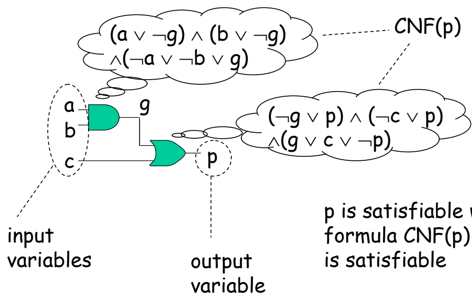
$M_i == M'_i, \forall i$?

$(M_i \oplus M'_i) == 0, \forall i$?

If out=1 is unsatisfiable, the two circuits are equivalent

From Circuit to AIG (CNF)

Can the circuit (maybe a PM) output be 1?



p is satisfiable when the formula CNF(p) ∧ p is satisfiable

Outline

- Symbolic Function Representation
- Combinational BDD-based verification
- Sequential BDD-based verification
- Satisfiability (SAT)
- Combinational Satisfiability-based verification
- Sequential Satisfiability-based verification

Bounded Model Checking (BMC)

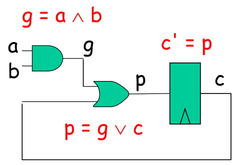
- Bounded Model Checking (Biere, et al., TACAS 1999)
  - Property checking method based on finite unfolding of transition relation interleaved with checks of the property
    - Sound: In its pure form no false positives are possible
    - Incomplete: Cannot guarantee correctness of property

Bounded Model Checking (BMC)

- Given
  - A finite transition system M
  - A property P (representing "good" states)
  - A non negative value *k* (bound)
- Create a SAT instance
  - Generate clauses for F_k (output a file in CNF format)
  - Call SAT on the CNF instance
  - A counterexample is a path from a state satisfying S_0 to state satisfying P, where every transition satisfies TR

Example

- Transition system described by a set of constraints



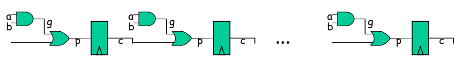
Model:

C = {
 g = a ∧ b,
 p = g ∨ c,
 c' = p
}

Each circuit element is a constraint
note: $a = a_t$ and $a' = a_{t+1}$

- Unfold the model k times

$U_k = TR_0 \wedge TR_1 \wedge \ldots \wedge TR_{k-1}$

S_0  P_k

- Use SAT solver to check satisfiability of

$S_0 \wedge U_k \wedge \neg P_k$

- A satisfying assignment is a counterexample of k steps

Basic Methods

- CNF-based
  - Use CNF-based SAT solver to represent unfolding and prove UNSAT for correctness of property
- Circuit-based
  - Use ATPG-like reasoning to show untestability
- Hybrid
  - Use circuit rewriting and SAT checking interleaved, e.g., based on AND/INV graphs

Applications

- Debugging
  - Can find counterexamples using a SAT solver
- Proving properties
  - Only possible if a bound on the length of the shortest counterexample is known
  - I.e., we need a *diameter* bound. The diameter is the maximum length of the shortest path between any two states
  - Worst case is exponential. Obtaining better bounds is sometimes possible, but generally intractable

Unbounded Model Checking (UMC)

- SAT and BMC can be also used for unbounded model checking
  - K-step induction
  - Abstraction
    - Counterexample-based
    - Non-counterexample-based
  - Exact image computations
    - SAT solver tests for fixed point
    - SAT solver computes image
  - Over-approximate image computations

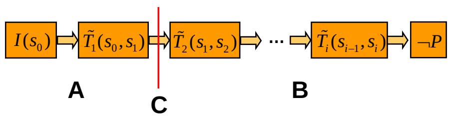
Interpolants (1/3)

- McMillan CAV 2002
- Given two Boolean functions *A* and B such that
  - $A \wedge B = 0$

  an interpolant is a function *C* such that:
  - $C \wedge B = 0$
  - $A \Rightarrow C$
  - *C* refers only to the common variables of *A* and B
- Interpolants can be easily computed from the refutation proof provided by SAT solvers

Interpolants (2/3)

- When performing a BMC check, we choose
  - $A = S_0 \wedge TR(S_0, S_1)$
  - $B = TR(S_1, S_2) \wedge \ldots \wedge TR(S_{k-1}, S_k) \wedge \neg P(S_k)$
- *Any interpolant provides an over-approximate image of the initial state S_0, guaranteed to be k-adequate w.r.t. $\neg P(S_k)$.*



Interpolants (3/3)

- Re-do BMC, replacing S_0 with the generated ITP, until intersection with $\neg P(S_k)$ or fix-point found
- In case of intersection, increase *k* and re-run
- It can be proved that *k* is bounded to the system diameter