Design Verification

Lecture 05 - Multi-Level Logic Verification II

1. Technique #5: Implication-based verification
   - Compute direct and indirect implications for each gate in the circuit
   - Check if the miter output = 1 is achievable

   Example 1: Direct implications

2. Implication graph
   - Given a circuit with n gates, there are $2n$ implication nodes
   - Each implication node indicates [g, v]
   - Edges in the graph indicate implication relations
   - Implications are transitive

   Example 2:

3. Contrapositive Law:
4. Constants

5. Indirect implications
   (a) Initialize implication graph with direct implications
   (b) for each node compute additional indirect implications
   (c) Key: learn as many implications as possible

Indirect_imply()
    for each node a in impl_graph
        S = transitive_closure(a);   /* all implications of a */
        assign values in S onto circuit;
        logic_simulate(S);
        /* any new values obtained during simulation is a new implication */
        add $a \rightarrow$ *new val* onto implication graph;
        add contrapositive if it does not exist;

Example 3:

6. Backward implications

7. Extended Backward implications

8. Recursive Learning
   - for all unjustified nodes in the implication list, repeatedly compute their backward implications
   - recursion depth: a parameter set by user

9. Maintaining small implication graph
   - Computing transitive closure depends on # edges & # nodes
   - Remove any redundant edges
   - Remove any redundant nodes
   - Cost of depth-first search reduced

10. Remove transitive edges

A simple Transitive Reduction Algorithm:

```
Transitive_reduce()
    for each node a in implication graph
        L = list of nodes directly reachable from a; /* 1 edge connecting them*/
        for each node b in L
            S = transitive_closure(b);
            for all nodes c in S
                if (c ∈ L)
                    remove edge (a, c) and remove c from L;
```

11. Eliminate equivalent nodes

$\longrightarrow$ Identify strongly-connected components (SCCs): every pair of nodes in a SCC have paths between them
$\longrightarrow$ By eliminating nodes in SCC (only 1 representative node necessary in the graph), we can eliminate 30% to 50% of all nodes in the implication graph

A simple SCC Algorithm:

12. Application to equivalence checking:

Example 4:

Example 4 continued