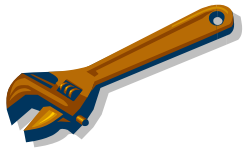
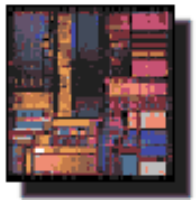






ECE 595Z
Digital Logic Systems Design Automation
Module 3 (Lectures 6-9): Two-level Logic Synthesis
Lecture 9



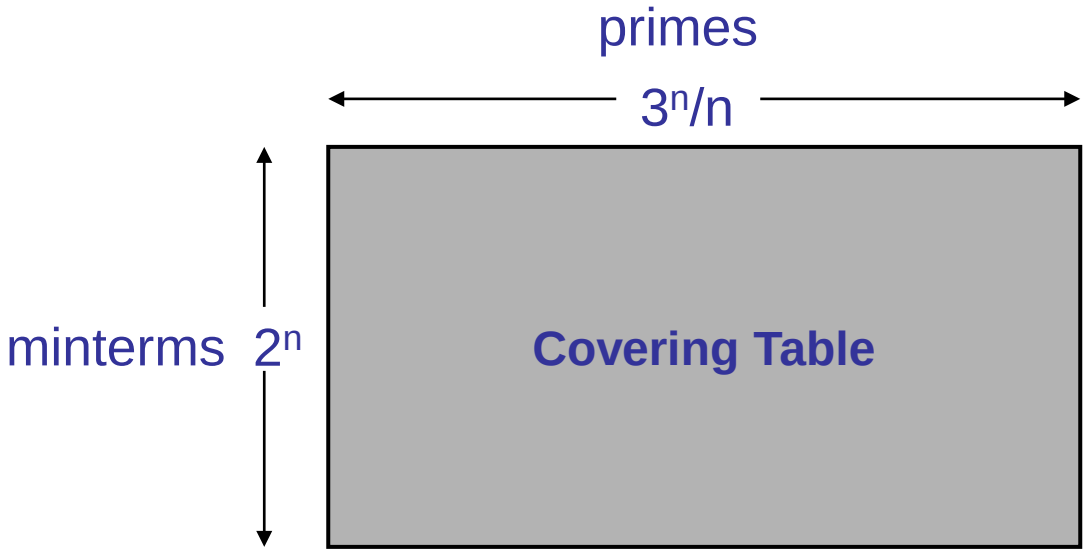
Anand Raghunathan

MSEE 348

raghunathan@purdue.edu

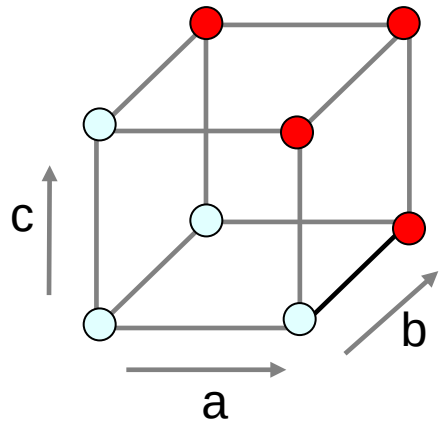
Quine-McCluskey: Scaling Challenges

- No. of primes *may* be large (worst case $3^n/n$)
- Covering problem is NP-complete, so exact algorithms *may* take exponential time in the worst case
- No. of minterms is *likely* to be large (2^n)

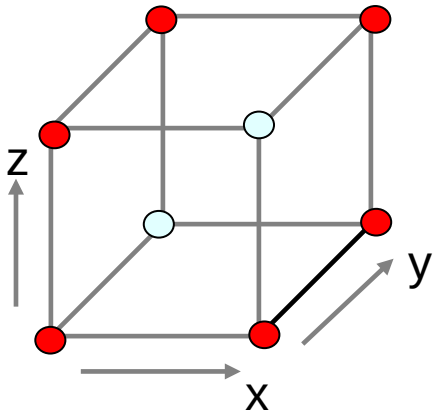


Unate Functions : Recap

- A logic function f is *positive unate* in variable x, if increasing x from 0 to 1 cannot decrease f from 1 to 0.
- A logic function f is *negative unate* in variable x, if increasing x from 0 to 1 cannot increase f from 0 to 1.
- A function *is unate* if it is unate in all its variables.
- A function that is not unate is *binate*



f is _____ in a, _____ in b, _____ in c



g is _____ in x, _____ in y, _____ in z

Unate Covers

- A *cover* F is *positive unate* in x_i iff $x_i' \notin c_j$ for all cubes $c_j \in F$
  - No 0's in position i for any cube in the cover
- A cover F is *negative unate* in x_i iff $x_i \notin c_j$ for all cubes $c_j \in F$
  - No 1's in position i for any cube in the cover
- A cover *is unate* if it is unate in all its variables.

F = {ab, a'c', bc'}

a	b	c
1	1	–
0	–	0
–	1	0

F is _____ in a, _____ in b, _____ in c

Unate Functions == Unate Covers?

Theorem: If a unate cover F exists for function f, then f is unate.

- A unate function may have a non-unate cover.

Example:

Results for Unate Functions

Theorem: The complement of a unate function is unate.

Theorem: The cofactors of a unate function with respect to any variable x and x' are unate.

Theorem: A unate cover is a tautology if and only if it contains a row of all –'s.

Theorem: A prime cover of a unate function is unate.

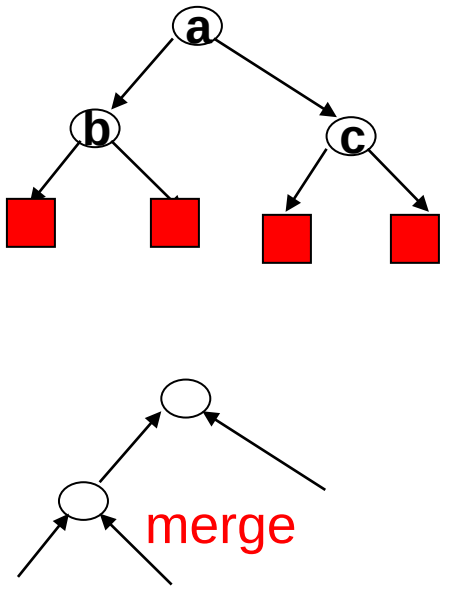
Theorem: Every prime of a unate function is essential.

Theorem: If a unate cover is SCC (single cube containment) minimal, it is a prime cover.

- Many operations are greatly simplified for unate functions
  - Tautology check, Prime generation, Complement, …

Unate Recursive Paradigm (URP)

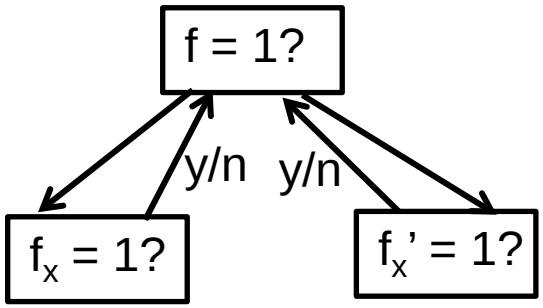
- What about binate functions?
  - Many functions in practice are binate
- Use Shannon's decomposition to recursively divide-and-conquer binate functions
  - Stop at Unate "leaves" and apply efficient techniques to perform desired operation
  - Merge results into desired result for original function



Main idea: *Operation* on unate leaf is computationally less complex

URP for Tautology Checking

- Input: Cover representing f
- Output: Yes/No (is f a tautology or not?)
- **Theorem**: f is a tautology iff f_x and $f_{x'}$ are both tautologies
- **Theorem**: A unate cover is a tautology iff it contains a tautology cube, i.e., row of all –'s.
- Suggests a recursive (divide-and-conquer) approach for tautology checking
- What do we need to know?
  - How to split (which variable to select?)
  - How to decide when to stop splitting?
  - How to merge the results?



URP for Tautology Checking

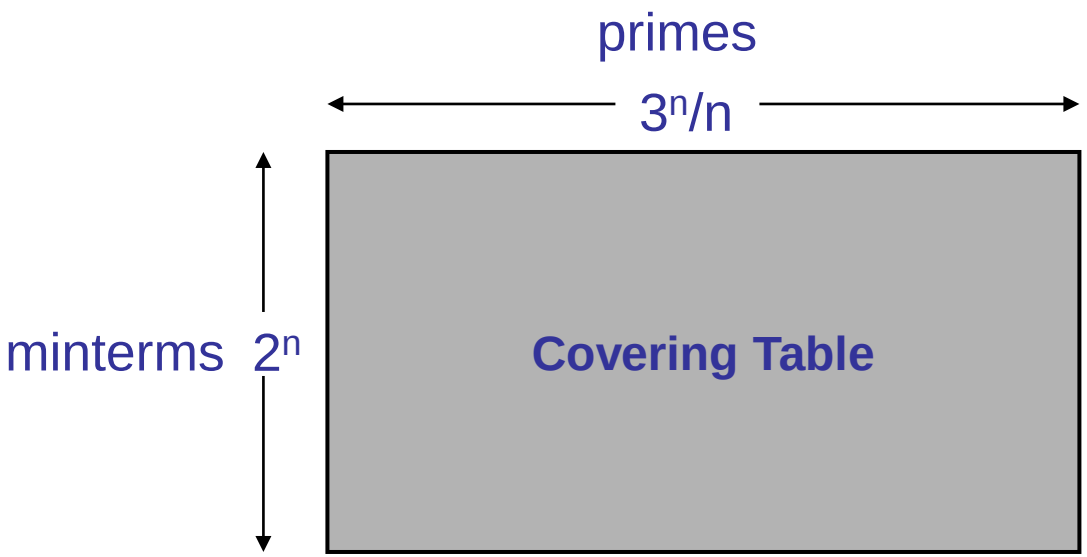
- Example F = {ab, ac, ab'c', a'}

General Principle

- Look for special cases of a problem that can be solved efficiently, try to reduce other cases to them.

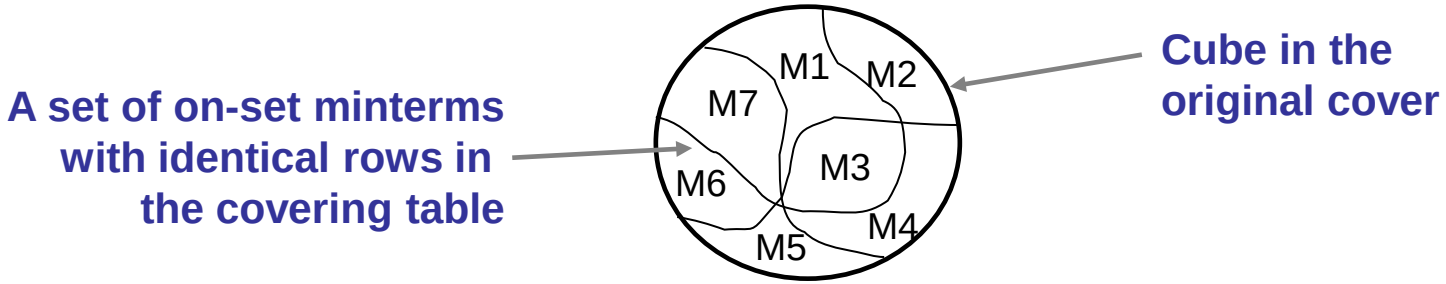
A More Complex Application of the Unate Recursive Paradigm

- Recall the challenges with the exact two-level minimization techniques that we have discussed?
  - No. of primes *may* be large (worst case $3^n/n$)
  - Covering problem is NP-complete, so the heuristics *may* take exponential time in the worst case
  - **No. of minterms is *likely* to be large (2^n)**



Can we Reduce the Size of the Covering Table by Construction?

- **Observation**: In practice, there are usually several sets of identical rows.
  - Each of these represents a set of minterms covered by the same set of primes.
  - If only we could just construct one row for each set to start with (rather than construct a large table and then try to reduce it)
- **Idea**: Start with each cube in a cover for F, and break it down into groups of minterms that will have identical rows in the covering table



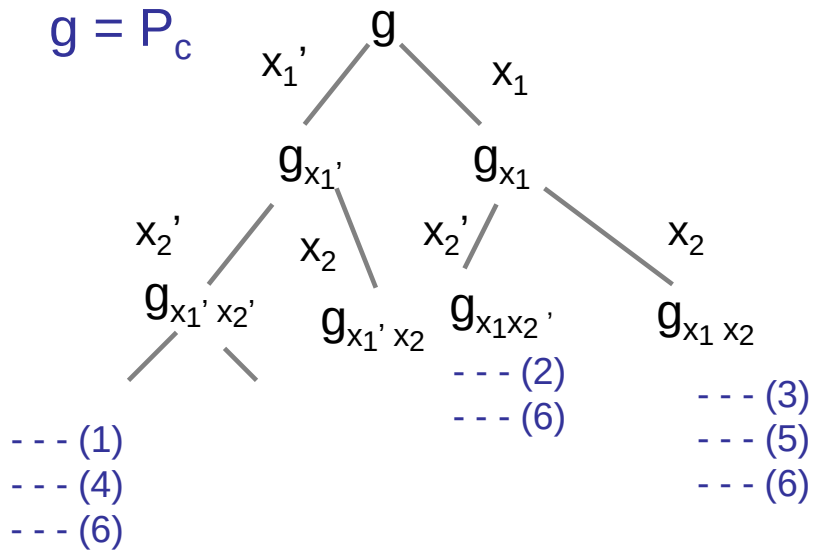
Covering Table Compression

Theorem: A cover C covers a cube c if and only if $C_c \equiv 1$.

- Now, let P be the set of all primes, and let c be a cube in the original function.
- Clearly $P_c \equiv 1$.
  - We just need to investigate which combinations of these primes is responsible for this tautology, i.e. for covering this cube.
  - Construct the covering table as a by-product of this tautology check

Covering Table Compression

- Build the recursive tautology tree for P_c.
  - Tag the primes that are left at any node.
  - Stop when you get a cover that has ALL rows of all –'s
  - Each leaf of this tree is a row of the compressed covering table.
  - The tags of the primes at the leaf is the set of primes that covers the row in the covering table.



Covering Table for c

123456
100101
…
010001
001011

→ Each row corresponds to a leaf of the tautology tree

- Repeat for each cube in the cover

Summary : Exact Two-level Minimization

- Generating primes
  - Tabular method
  - Iterated consensus
- Generating a compressed covering table
  - Unate Recursive Paradigm
- Solving the cyclic core
  - Petrick's method
  - MIN-SAT
  - Branch and bound
    - Maximal independent set

Heuristic Two-Level Minimization

- Exact minimization (Quine McCluskey) often too expensive
- Approximate (heuristic) techniques don't guarantee the optimal solution, but strive for close to optimal results.
- We will focus on ESPRESSO-II, a two-level minimizer developed at U.C. Berkeley
  - Richard Rudell, Robert Brayton, and Alberto Sangiovanni-Vincentelli

Key Developments in Heuristic Two-level Minimization

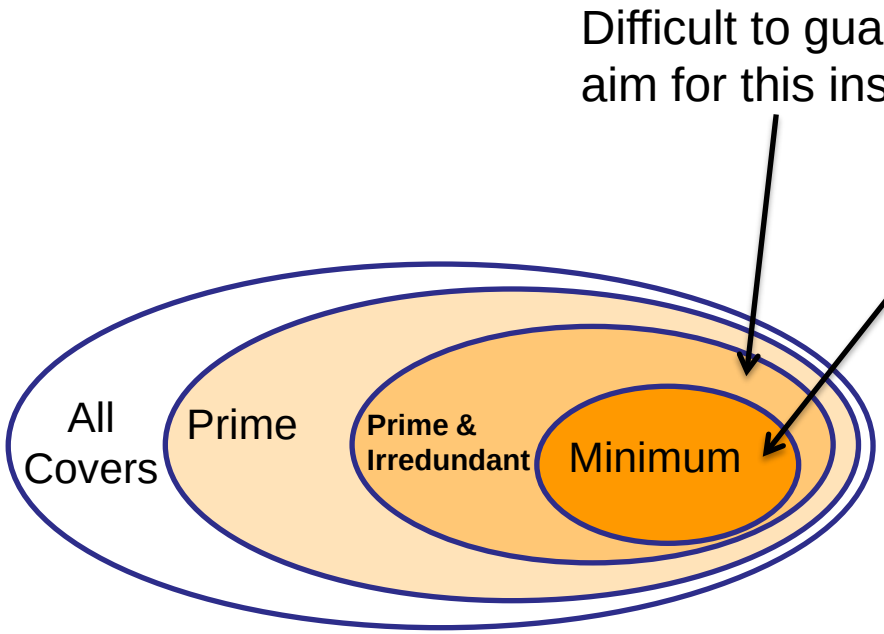
- MINI logic minimizer developed at IBM in the mid-1970's
  - S. J. Hong, R. G. Cain, and D. L. Ostapko, "MINI: A Heuristic Approach for Logic Minimization", IBM Journal of R&D, Sept. 1974.
- PRESTO logic minimizer developed at Tektronix in late 1970's
  - D. W. Brown, "A State-Machine Synthesizer -- SMS", Design Automation Conference, pp. 301-304, June 1981.
- ESPRESSO-I started off as an effort to implement these algorithms in a unified SW framework
  - R. K. Brayton, G. D. Hachtel, L. Hemachandra, A. R. Newton and A. L. Sangiovanni-Vincentelli, "A Comparison of Logic Minimization Strategies Using ESPRESSO: An APL Program Package for Partitioned Logic Minimization", Int. Symp. On Circuits and Systems, pp. 42-48, May 1982.

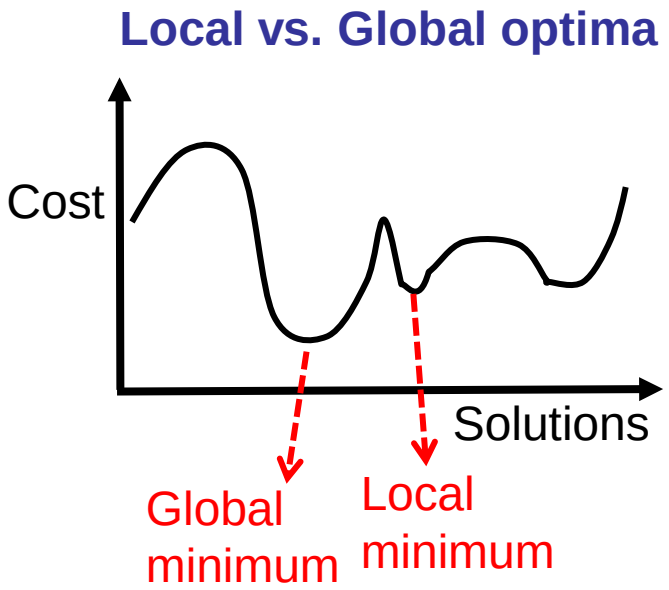
Key Developments in Heuristic Two-level Minimization

- Insights gained during implementation of ESPRESSO-I led to development of improved algorithms and ESPRESSO-II
  - R. K. Brayton, G. D. Hachtel, C. D. McMullen, and A. L. Sangiovanni-Vincentelli, "Logic Minimization Algorithms for VLSI", Kluwer Academic Publishers, 1984.
- Implemented by Richard L. Rudell for his M.S. thesis project at U. C. Berkeley
  - Extended to multi-valued minimization

Degrees of Optimality in Two-level Minimization

- Cover > Prime Cover > Irredundant Prime Cover > Minimum Prime Cover

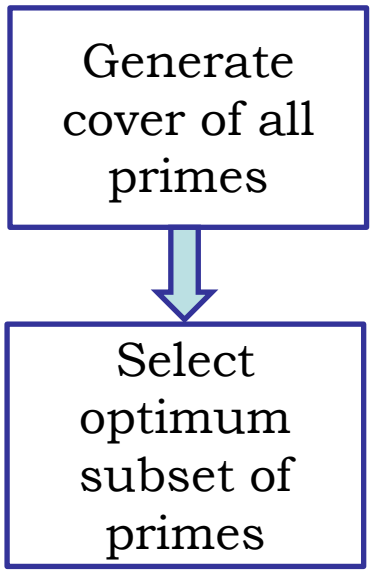




Strategy: Iterative Improvement

- Use weaker optimization steps, but iterate to obtain successive improvements

Exact Two-level Minimization



Approximate Two-level Minimization