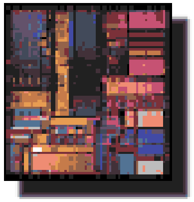
ECE 595Z
Digital Logic Systems Design Automation

Module 3 (Lectures 6-9): Two-level Logic Synthesis
Lecture 8



Anand Raghunathan

MSEE 348

raghunathan@purdue.edu

Lecture #8 Outline

- Wrapup: Selecting a subset of primes
- Generating primes
- Scaling the QM algorithm

Putting it Together : Branch and Bound with MIS Computation

```
BB(T, best_soln, current_soln) {
        Reduce(T, current_soln);
        if(T is empty) {
            if(cost(current_soln) < cost(best_soln) ) {
                best_soln = current_soln;
                return(best_soln);
            } else {
                return(NULL);
            }
        }
        L = compute_MIS_size(T);
        if(L + cost(current_soln)  ≥ cost(best_soln) ) {
            return(NULL);
        }
        j = choose_column(T);
        soln1 = BB(T, best_soln, current_soln ∪ j);
        if(cost(soln1) == L ) return(soln1);
        soln0 = BB(T – j, best_soln, current_soln);
        return(lower_cost(soln1, soln0));
}
```

Remove empty rows and columns, apply pruning techniques (essential, equivalence & dominance)

Reached leaf of search tree. Keep solution if it is better than the best seen thus far.

Evaluate bounding criterion and discard part of the search tree if possible.

Branching variable

Recursive calls to explore two cases – either the column is selected or it is not

Branch and Bound with MIS Computation : Examples

Example 1

	p1	p2	p3	p4	p5	p6
1	1					1
2	1	1				
3		1	1			
4			1	1		
5				1	1	
6					1	1

Example 2

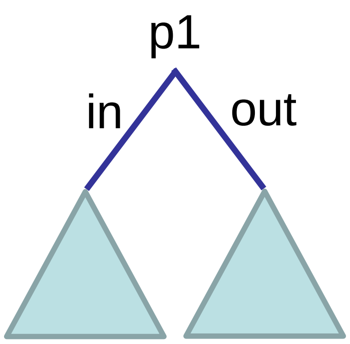
	p1	p2	p3	p4	p5	p6
1	1				1	
2	1	1		1		
3		1	1			
4				1	1	1
5			1	1		
6		1				1
7	1		1			

Question

- Recall the CNF formula representation of the covering table used in Petrick's method?
- What is the equivalent of branching in the branch and bound algorithm
  - Think in terms of applying operations that we have learnt on the CNF formula

Example:

	p1	p2	p3	p4	p5	p6
	a'c	b'c	a'b	bc'	ab'	ac'
m3	1	1				
m5			1	1		
m7	1		1			
m9					1	1
m11		1			1	
m13				1		1



P = (P1+P2)(P3+P4)(P1+P3)(P5+P6)(P2+P5)(P4+P6)

Branch p1 in

Branch p3 in

Using Iterative Independent Sets for Heuristic Prime Selection

Let $\boldsymbol{I} = \{ I_1, I_2 \ldots I_k \}$ be an independent set of rows

1. Choose column j which covers $I_i \in \boldsymbol{I}$ and the most rows of T.
2. Put $j \rightarrow J$ (set of columns in the cover)
3. Eliminate all rows covered by column j
4. $\boldsymbol{I} \leftarrow \boldsymbol{I} - I_i$
5. Go to 1 if $| \boldsymbol{I} | > 0$
6. If T is empty, then done

   (if this is reached after processing the first independent set, we have the guaranteed *minimum solution*)
7. If T is not empty, choose a new independent set of T and go to 1

- Sub-optimal in general

Summary : Selecting a Subset of Primes

- Four approaches discussed in class

Approach	Advantages	Disadvantages
Petrick's method	Simple, exact.	Generates all solutions. Very likely to be exponential.
MIN-SAT	Exact. Leverage advances in SAT solvers.	Solvers may not exploit full knowledge of the covering problem. Exponential in the worst case.
Branch and Bound	Exact. Incorporate problem knowledge through branching and bounding heuristics.	Exponential in the worst case.
Iterative Independent Sets	Polynomial time (if approximate MIS algorithm is used)	Sub-optimal.

Can you think of any improved heuristics?

QM : Generation of Primes

- Need to generate all primes ☹
  - Tabular Method
  - Iterated consensus

Generating Primes : Tabular Method

- Start with minterm canonical form of F
- Group *pairs* of adjacent minterms into cubes
- Repeat merging of cubes until no more merging possible; mark (√) and remove all covered cubes.
- Result: set of *primes* of f.

Example:

$F = x'y' + wxy + x'yz' + wy'z$

0	$w'x'y'z'$		
1	$w'x'y'z$ $w'x'yz'$ $wx'y'z'$		
2	$wx'y'z$ $wx'yz'$		
3	$wxyz'$ $wxy'z$		
4	$wxyz$		

Generating Primes : Tabular Method

$$F = x'y' + wxy + x'yz' + wy'z$$

w’ x’ y’ z’ √	w’ x’ y’ √ w’ x’ z’ √ x’ y’ z’ √	x’ y’ x’ z’
w’ x’ y’ z √ w’ x’ y z’ √ w x’ y’ z’ √	x’ y’ z √ x’ y z’ √ w x’ y’ √ w x’ z’ √	
w x’ y’ z √ w x’ y z’ √	w y’ z w y z’	
w x y z’ √ w x y’ z √	w x y w x z	
w x y z √		

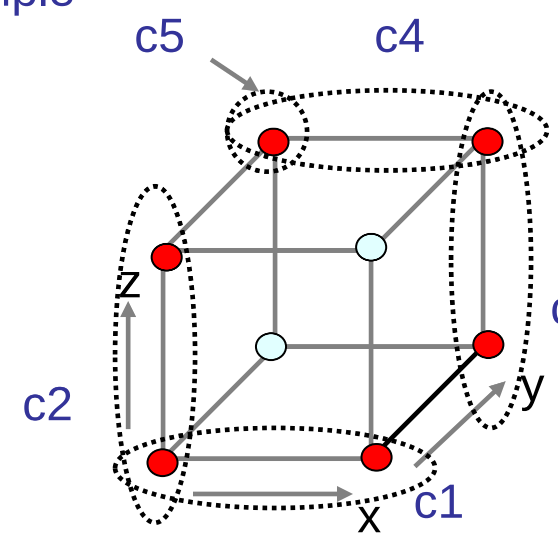
Iterated Consensus : Motivation

- Problems with Tabular Method
  - Need to start off with all minterms
    - *Likely* to be exponential
- Better approach
  - Given a cover for F (set of cubes), generate the set of all primes
  - A technique called **iterated consensus** can be used for this purpose

Distance Between Cubes

- The distance, δ, between two cubes is the number of dimensions the cubes differ on.
  - Differ : one cube has a "1" and the other has a "0"
  - δ is the number of ϕ entries in a bitwise intersection of the cube vectors.

Example



Vector representation of cubes:
c1 = [- 0 0]; c2 = [0 0 -];
c3 = [1 1 -]; c4 = [- 1 1]; c5 = [0 1 1]

δ(c1,c2) =
δ(c1,c3) =
δ(c1,c4) =
δ(c4,c5) =

Consensus

- The consensus between two cubes, c1 and c2 is defined as:
  - If $\delta(c1, c2) > 1$

    $c1 \odot c2 = \phi$
  - If $\delta(c1, c2) = 0$

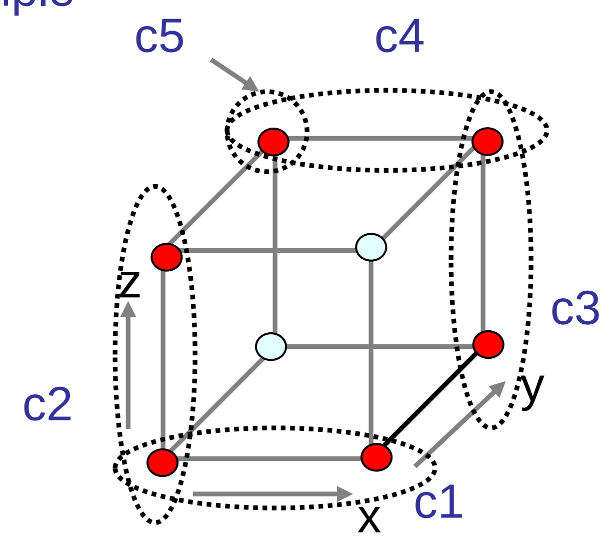
    $c1 \odot c2 = c1 \cap c2$
  - If $\delta(c1, c2) = 1$
    - If $c1[k] \cap c2[k] \neq \phi$

      $(c1 \odot c2)[k] =$

      $c1[k] \cap c2[k]$
    - If $c1[k] \cap c2[k] = \phi$

      $(c1 \odot c2)[k] = -$

Example



Vector representation of cubes:
c1 = [- 0 0]; c2 = [0 0 -];
c3 = [1 1 -]; c4 = [- 1 1]; c5 = [0 1 1]

$c1 \odot c2 =$
$c1 \odot c3 =$
$c1 \odot c4 =$

Iterated Consensus

- Starts with a cover and generates new implicants through consensus till no more implicants can be generated.

```
iterated_consensus(C) {
    do {
       foreach (ci, cj ε C) {
         C = C ∪ (ci ⊙ cj);
         C = SCC_minimal(C);
       }
    } until (no change in C);
}
```

- `SCC_minimal` checks for single cube containment
  - If a cube c1 is contained in some other cube c2 in C, then c1 is deleted.

Theorem [Quine]: Iterated Consensus generates all primes of a function.

Iterated Consensus : Example

```
iterated_consensus(C) {
    do {
        foreach (ci, cj ε C) {
            C = C ∪ (ci ⦿ cj);
            C = SCC_minimal(C);
        }
    } until (no change in C);
}
```

Example:

$F(x_1,x_2,x_3,x_4) = \{x_1x_2,\ x_2'x_3,\ x_2x_3x_4\}$

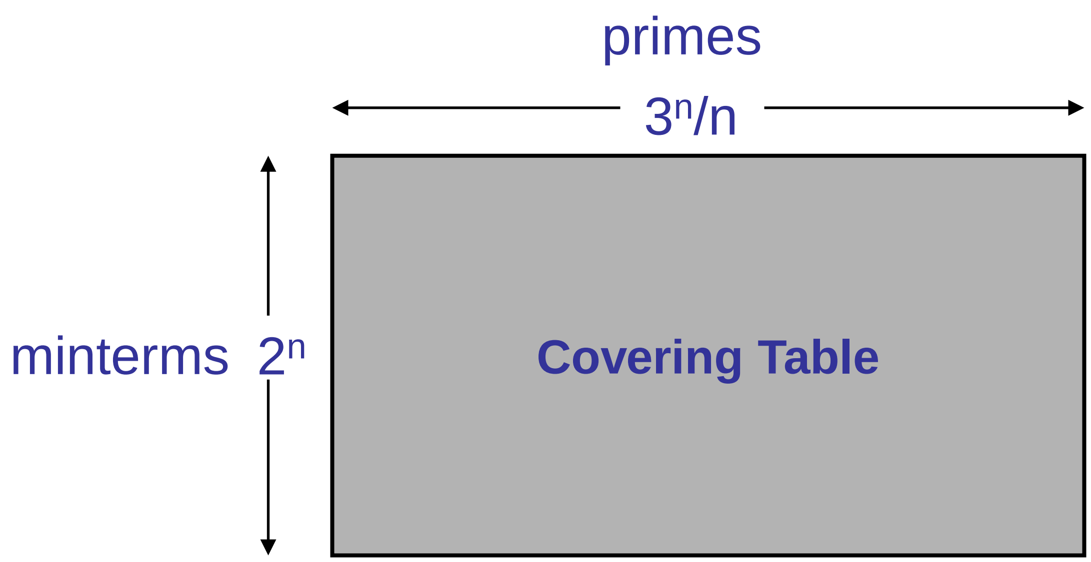
c_1 = [11--]
c_2 = [-01-]
c_3 = [-111]

Summary : Exact two-level minimization

- Solving the cyclic core
  - Petrick's method
  - MIN-SAT
  - Branch and bound
    - Maximal independent set
  - Iterative independent set
- Generating primes
  - Tabular method
  - Iterated consensus

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)



Required Reading: Two-level minimization

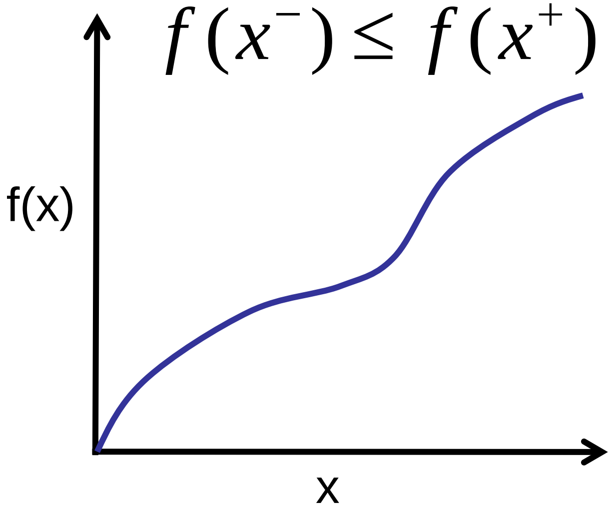
- Richard Rudell's PhD thesis, Chapter 2 (pages 9-31)
  - Posted on blackboard
  - Will be helpful in reinforcing the concepts we speak about in class

Unate Functions

- Analogy to integer and real-valued functions
  - *Monotone functions*
- 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.

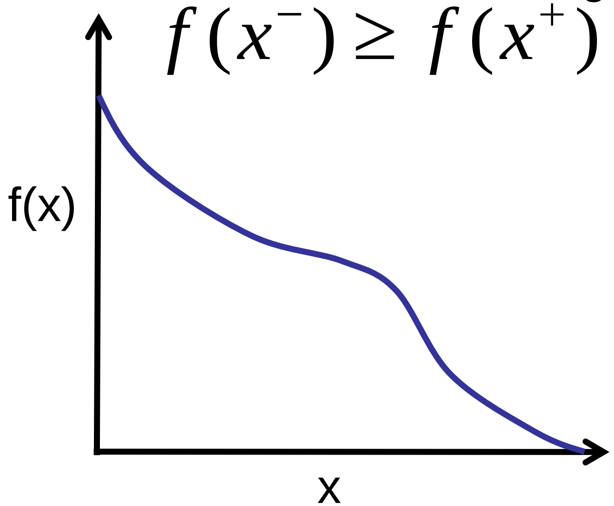
Monotone increasing

$$f(x^-) \le f(x^+)$$



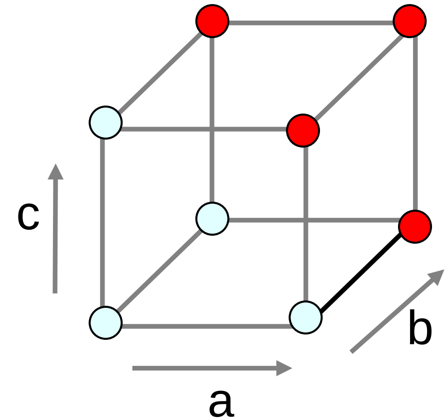
Monotone decreasing

$$f(x^-) \ge f(x^+)$$



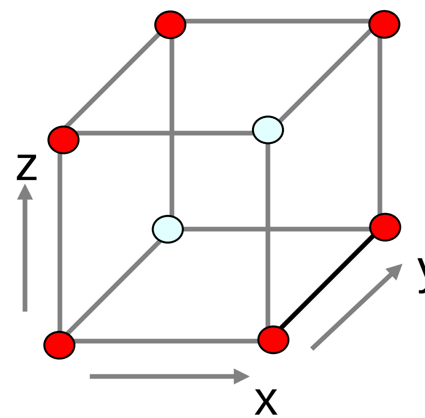
Unate Functions : Examples

- 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(a,b,c)

f is ________ in a, ________ in b, ________ in c



g(x,y,z)

g is ________ in x, ________ in y, ________ in z