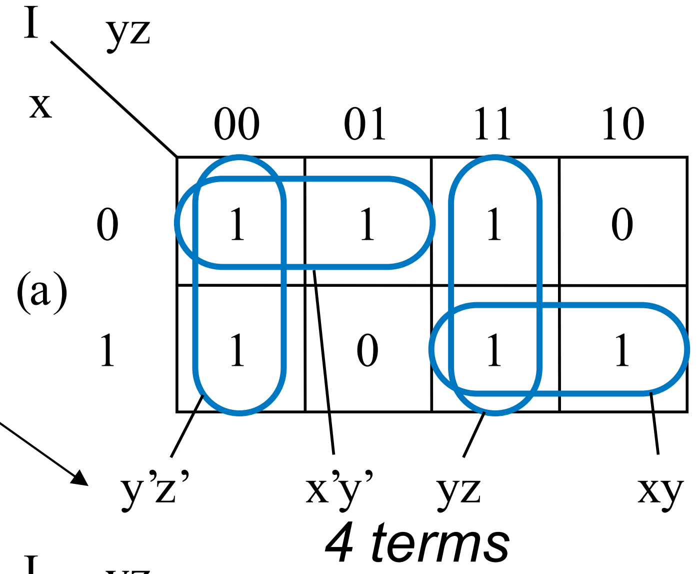
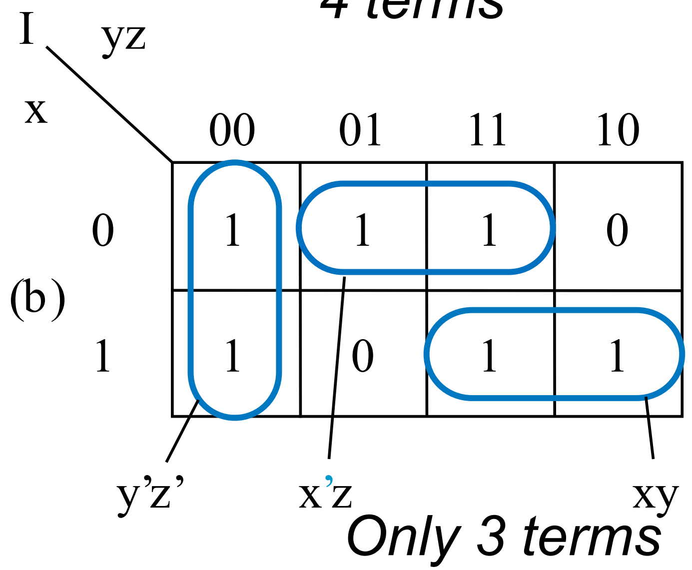
Automating Two-Level Logic Size Optimization

- Minimizing by hand
  - Is hard for functions with 5 or more variables
  - May not yield minimum cover depending on order we choose
  - Is error prone
- Minimization thus typically done by automated tools
  - *Exact algorithm*: finds optimal solution
  - *Heuristic*: finds good solution, but not necessarily optimal

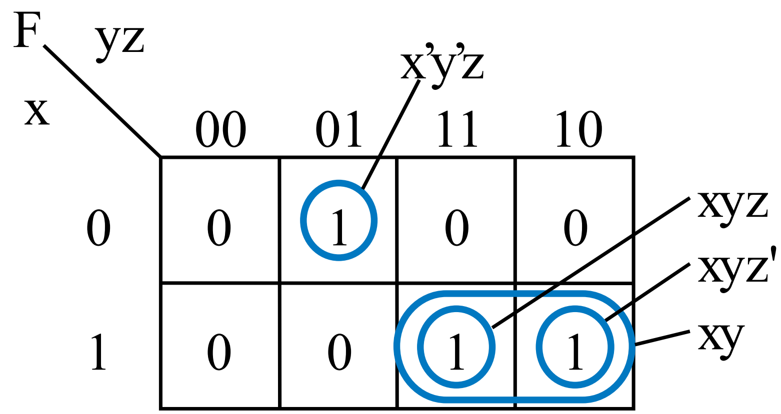
a





Basic Concepts Underlying Automated Two-Level Logic Size Optimization

- Definitions
  - *On-set*: All minterms that define when F=1
  - *Off-set*: All minterms that define when F=0
  - *Implicant*: Any product term (minterm or other) that when 1 causes F=1
    - On K-map, any legal (but not necessarily largest) circle
    - Cover: Implicant xy *covers* minterms xyz and xyz'
  - *Expanding* a term: removing a variable (like larger K-map circle)
    - xyz → xy is an expansion of xyz



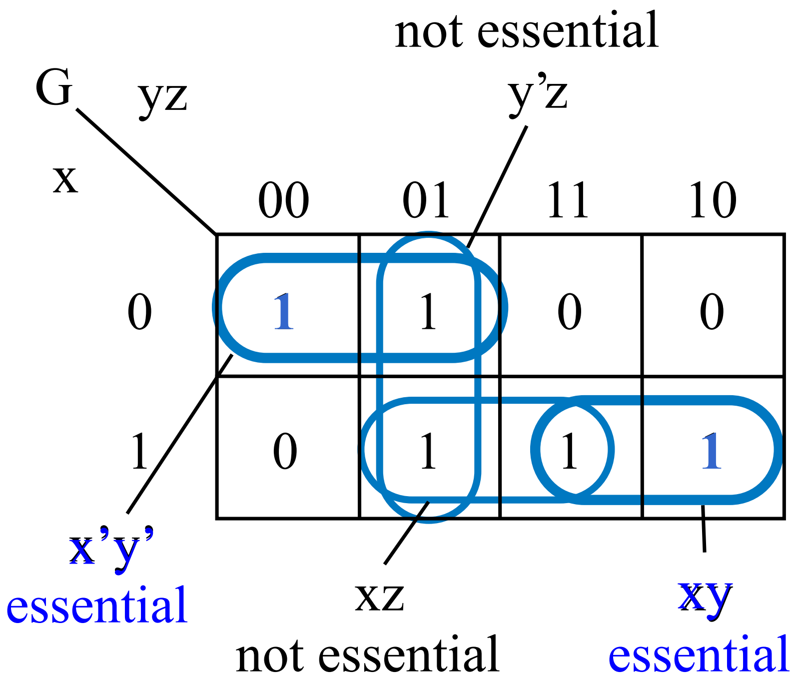
4 implicants of F

*Note: We use K-maps here just for intuitive illustration of concepts; automated tools do **not** use K-maps.*

- ***Prime implicant***: Maximally expanded implicant – any expansion would cover 1s not in on-set
  - x'y'z, and xy, above
  - But not xyz or xyz' – they can be expanded

Basic Concepts Underlying Automated Two-Level Logic Size Optimization

- Definitions (cont)
  - *Essential prime implicant*: The only prime implicant that covers a particular minterm in a function's on-set
    - Importance: We ***must*** include ***all*** essential PIs in a function's cover
    - In contrast, some, but not all, non-essential PIs will be included



Automated Two-Level Logic Size Optimization Method

TABLE 6.1 Automatable tabular method for two-level logic size optimization.

	Step	Description
1	*Determine prime implicants*	Starting with minterm implicants, methodically compare all pairs (actually, all pairs whose numbers of uncomplemented literals differ by one) to find opportunities to combine terms to eliminate a variable, yielding new implicants with one less literal. Repeat for new implicants. Stop when no implicants can be combined. All implicants not covered by a new implicant are prime implicants.
2	*Add essential prime implicants to the function's cover*	Find every minterm covered by only one prime implicant, and denote that prime implicant as essential. Add essential prime implicants to the cover, and mark all minterms covered by those implicants as already covered.
3	*Cover remaining minterms with nonessential prime implicants*	Cover the remaining minterms using the minimal number of remaining prime implicants.

- Steps 1 and 2 are exact
- Step 3: Hard. Checking all possibilities: exact, but computationally expensive. Checking some but not all: heuristic.

Quine-McCluskey method

- Implicant (항 또는 항목) : SOP 표현에서 해당 항목이 1일때 출력함수가 1인 항
- Prime implicant (주항 또는 주항목): 논리식을 간단하게 했을 때 더 이상 간단히 되지 않는 항목
- Essential prime implicant (필수 주항): 함수의 완전한 표현을 위해 반드시 포함되어야 하는 주항목
- Quine-McCluskey method
  - K-map과 동일한 과정을 통해 논리식을 간단하게 만드는 알고리즘
  - K-map과 동일한 연산을 표 기반으로 수행하여 소프트웨어적으로 구현하기 용이함

Quine-McCluskey method

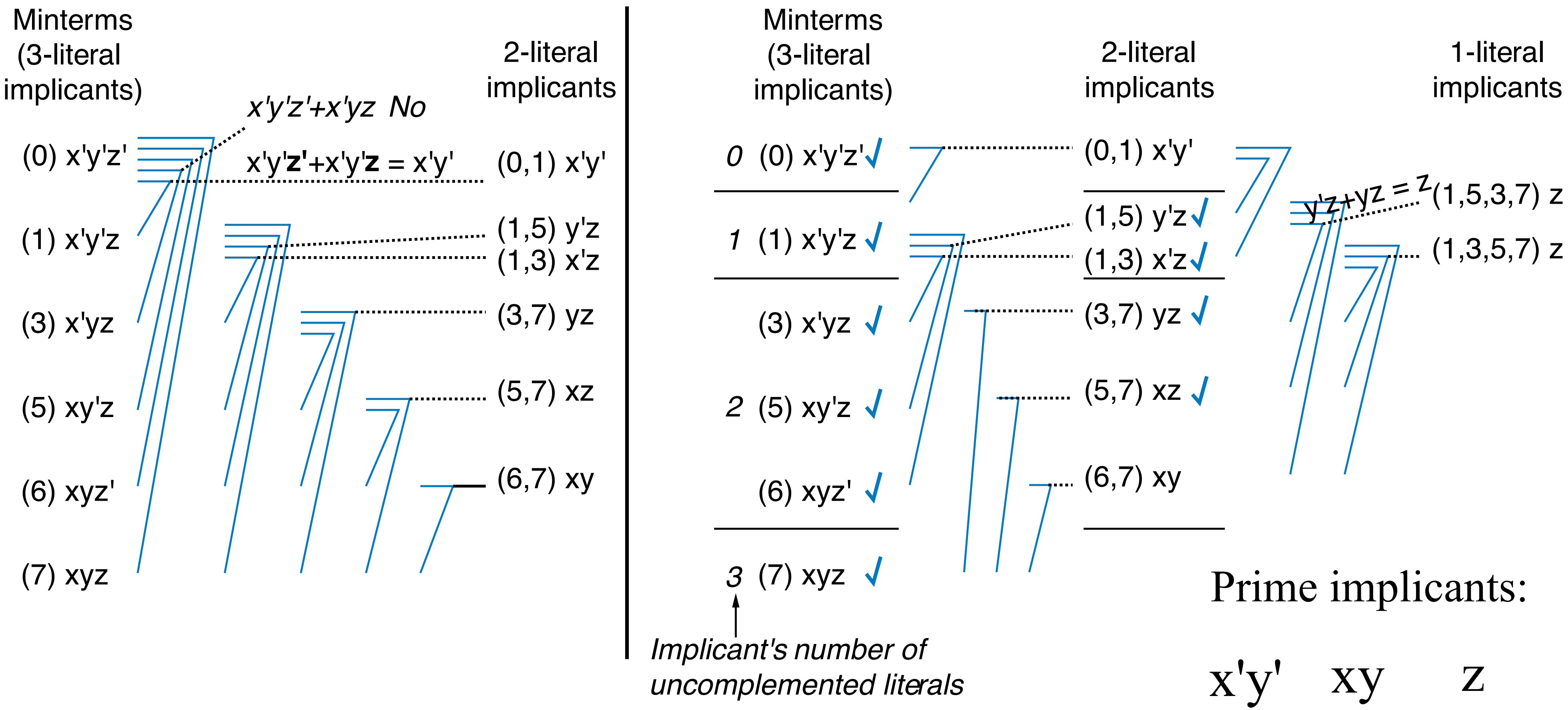
- 1. Generate Prime Implicants
  - K-map에서 box를 키우는 과정에 해당
  - Exhaustive method: 모든 경우를 비교하여 수행
    - 완벽 (Complete) 하지만 시간이 오래 걸림 ($3^n/n$)
  - Iterative method: 모든 경우를 비교하여 수행
    - 입력된 SOP 항을 점차 확장시켜가며 수행
    - 1개의 변수가 다른 경우를 비교 (K-map에서 옆 항으로 확장하는 경우에 해당)
    - 확장의 방식은 heuristic 에 의해 결정

Tabular Method Step 1: Determine Prime Implicants

Methodically Compare All Implicant Pairs, Try to Combine

- Example function: F = x'y'z' + x'y'z + x'yz + xy'z + xyz' + xyz

Actually, comparing ALL pairs isn't necessary—just pairs differing in uncomplemented literals by one.



Prime implicants:

x'y' xy z

Problem with Methods that Enumerate all Minterms or Compute all Prime Implicants

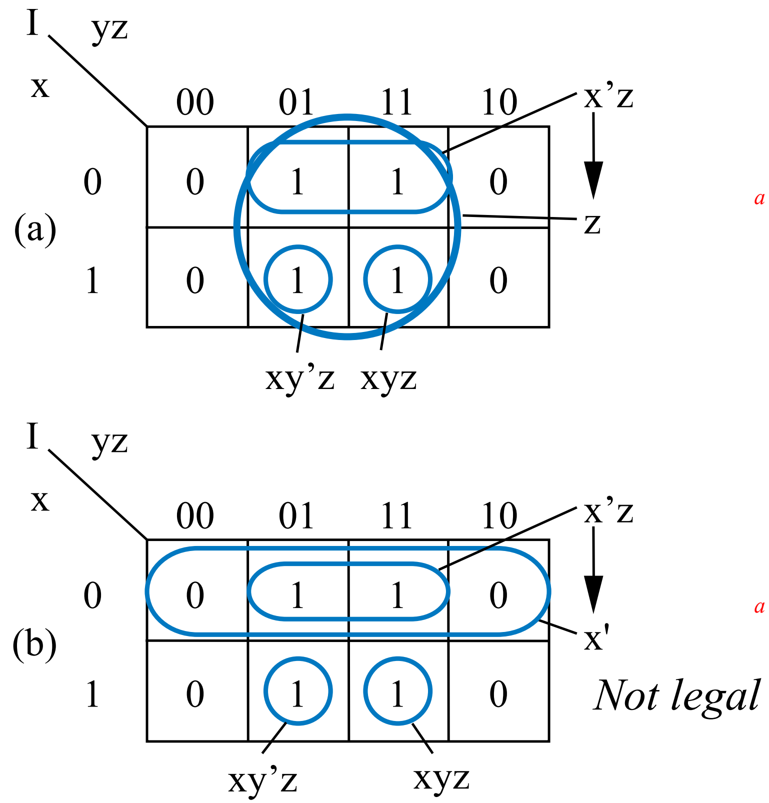
- Too many minterms for functions with many variables
  - Function with 32 variables:
    - 2^{32} = 4 billion possible minterms.
    - Too much compute time/memory
- Too many computations to generate all prime implicants
  - Comparing every minterm with every other minterm, for 32 variables, is $(4 \text{ billion})^2$ = 1 quadrillion computations
  - Functions with many variables could requires days, months, years, or more of computation – unreasonable

Solution to Computation Problem

- Solution
  - Don't generate all minterms or prime implicants
  - Instead, just take input equation, and try to "iteratively" improve it
  - Ex: F = abcdefgh + abcdefgh'+ jklmnop
    - Note: 15 variables, may have thousands of minterms
    - But can minimize just by combining first two terms:
      - F = abcdefg(h+h') + jklmnop = abcdefg + jklmnop

Two-Level Optimization using Iterative Method

- Method: Randomly apply “expand” operations, see if helps
  - Expand: remove a variable from a term
    - Like expanding circle size on K-map
      - e.g., Expanding x’z to z legal, but expanding x’z to z’ not legal, in shown function
      - After expand, remove other terms covered by newly expanded term
  - Keep trying (iterate) until doesn’t help



Ex:
F = abcdefgh + abcdefgh’+ jklmnop
F = abcdefg + abcdefgh’ + jklmnop
F = abcdefg + jklmnop

Covered by newly expanded term abcdefg

Illustrated above on K-map, but iterative method is intended for an automated solution (no K-map)

Ex: Iterative Hueristic for Two-Level Logic Size Optimization

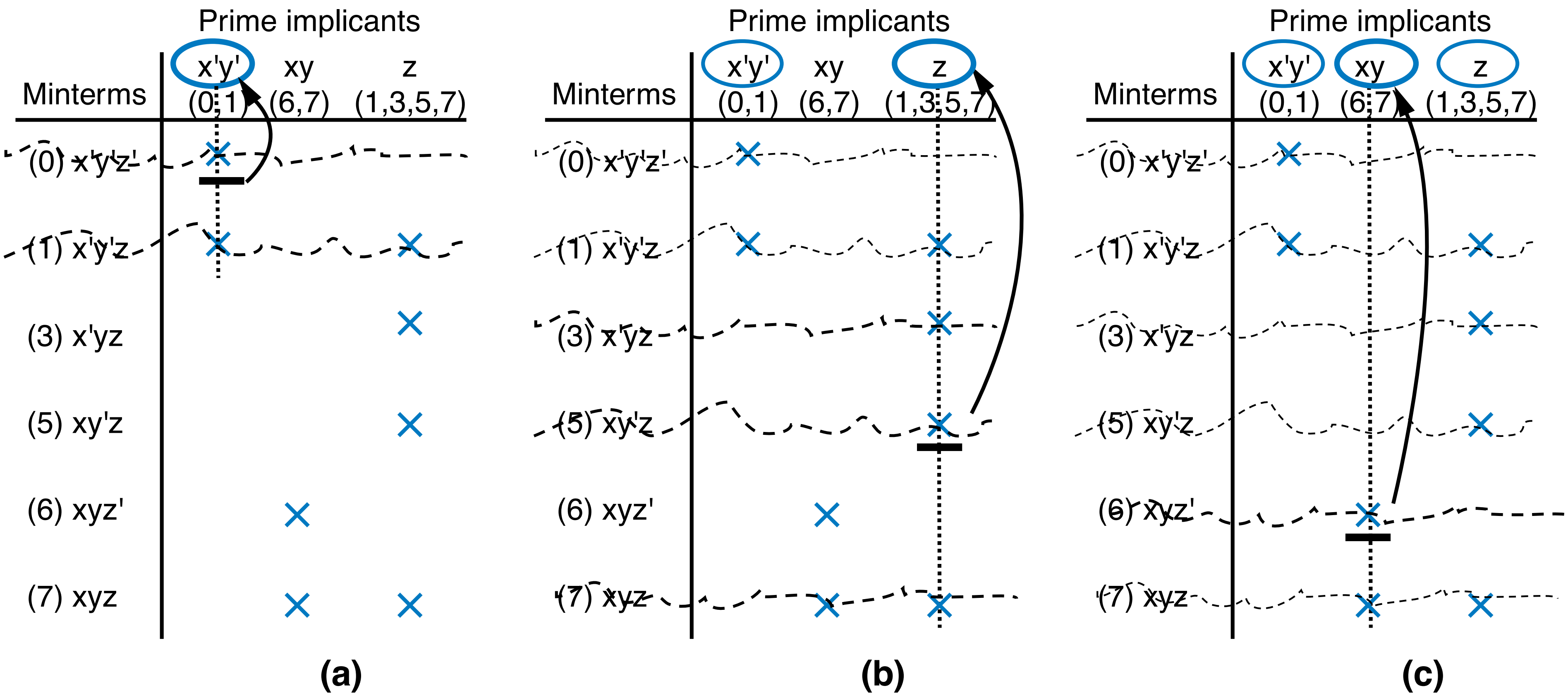
- F = xyz + xyz' + x'y'z' + x'y'z   *(minterms in on-set)*
- Random expand: F = xy~~z~~ + xyz' + x'y'z’ + x'y'z
  - Legal: Covers xyz' and xyz, both in on-set
  - Any implicant covered by xy? Yes, xyz'.
- F = xy + ~~xyz'~~ + x'y'z' + x'y'z
- Random expand: F = x~~y~~ + x'y'z' + x'y'z
  - Not legal (x covers xy'z', xy'z, xyz', xyz: two not in on-set)   a
- Random expand: F = xy + x'y'~~z'~~ + x'y'z
  - Legal
  - Implicant covered by x'y': x'y'z
- F = xy + x'y'z' + ~~x'y'z~~

Quine-McCluskey method

- 2. Construct Prime Implicant Table
  - 항과 주항을 행과 열로 가지는 표를 생성
- 3. Reduce Prime Implicant Table
  - (a) Remove Essential Prime Implicants
    - 어떤 항이 오직 하나의 PI에만 포함되는 경우 -> Essential PI
  - (b) Row Dominance
  - (c) Column Dominance
- 4. Solve Prime Implicant Table

Tabular Method Step 2: Add Essential PIs to Cover

- Prime implicants (from Step 1): x'y', xy, z



If only one **X** in row, then that PI is essential—it's the only PI that covers that row's minterm.

Tabular Method Step 3: Use Fewest Remaining PIs to Cover Remaining Minterms

- Essential PIs (from Step 2): x'y', xy, z
  - Covered all minterms, thus nothing to do in step 3
- Final minimized equation:

  F = x'y' + xy + z

	Prime implicants		
Minterms	x'y' (0,1)	xy (6,7)	z (1,3,5,7)
(0) x'y'z'	X		
(1) x'y'z	X		X
(3) x'yz			X
(5) xy'z			X
(6) xyz'		X	
(7) xyz		X	X

(c)

Quine-McCluskey method example

List Minterms

Column I	
0	0000
2	0010
8	1000
5	0101
6	0110
10	1010
12	1100
7	0111
13	1101
14	1110
15	1111

Step 1

Column I

0	0000	√
2	0010	√
8	1000	√
5	0101	√
6	0110	√
10	1010	√
12	1100	√
7	0111	√
13	1101	√
14	1110	√
15	1111	√

Column II

(0,2)	00-0	√
(0,8)	-000	√
(2,6)	0-10	√
(2,10)	-010	√
(8,10)	10-0	√
(8,12)	1-00	√
(5,7)	01-1	√
(5,13)	-101	√
(6,7)	011-	√
(6,14)	-110	√
(10,14)	1-10	√
(12,13)	110-	√
(12,14)	11-0	√
(7,15)	-111	√
(13,15)	11-1	√
(14,15)	111-	√

Column III

(0,2,8,10)	-0-0
(0,8,2,10)	-0-0
(2,6,10,14)	–10
(2,10,6,14)	–10
(8,10,12,14)	1–0
(8,12,10,14)	1–0
(5,7,13,15)	-1-1
(5,13,7,15)	-1-1
(6,7,14,15)	-11-
(6,14,7,15)	-11-
(12,13,14,15)	11–
(12,14,13,15)	11–

Quine-McCluskey method example

Step 2: Construct Prime Implicant Table.

	$B'D'$ (0,2,8,10)	CD' (2,6,10,14)	BD (5,7,13,15)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
0	X					
2	X	X				
5			X			
6		X		X		
7			X	X		
8	X				X	
10	X	X			X	
12					X	X
13			X			X
14		X		X	X	X
15			X	X		X

Quine-McCluskey method example

(i) Remove Primary Essential Prime Implicants

	$B'D'(*)$ (0,2,8,10)	CD' (2,6,10,14)	$BD(*)$ (5,7,13,15)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
$(\circ)0$	X					
2	X	X				
$(\circ)5$			X			
6		X		X		
7			X	X		
8	X				X	
10	X	X			X	
12					X	X
13			X			X
14		X		X	X	X
15			X	X		X

$*$ indicates an essential prime implicant

$\circ$ indicates a distinguished row, i.e. a row covered by only 1 prime implicant

Quine-McCluskey method example

- Primary essential prime implicants are identified.
  - These are implicants which will appear in any solution.
  - A row which is covered by only 1 prime implicant is called a distinguished row.
  - The prime implicant which covers it is an essential prime implicant.
  - In this step, essential prime implicants are identified and removed.
- The corresponding column is crossed out.
  - Also, each row where the column contains an X is completely crossed out, since these minterms are now covered.
  - These essential implicants will be added to the final solution. In this example, B'D' and BD are both primary essentials.

Quine-McCluskey method example

Row dominnace: A dominating row can always be eliminated.

	CD' (2,6,10,14)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
6	X	X		
12			X	X
14	X	X	X	X

Column dominnace: A dominated column can always be eliminated.

	CD' (2,6,10,14)	BC (6,7,14,15)	AD' (8,10,12,14)	AB (12,13,14,15)
6	X	X		
12			X	X

Quine-McCluskey method example

Iteration #2.

(i) Remove Secondary Essential Prime Implicants

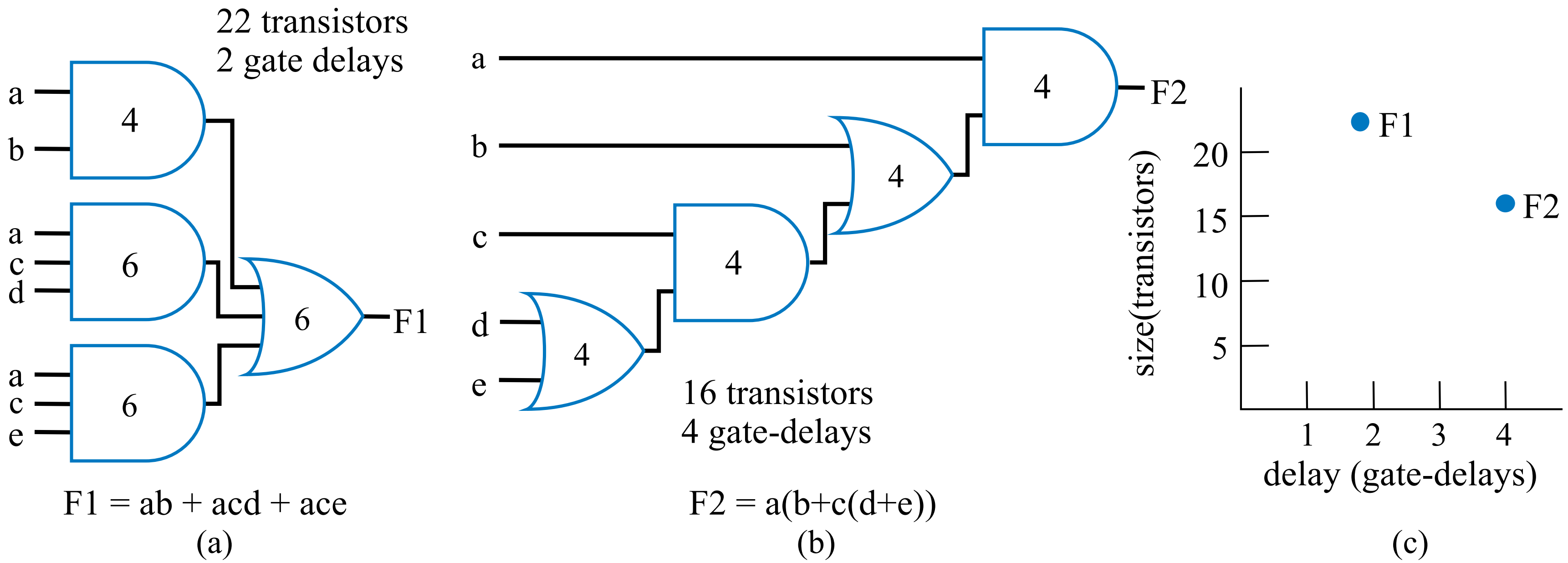
	$CD'(**)$ (2,6,10,14)	$AD'(**)$ (8,10,12,14)
$(\circ)6$	X	
$(\circ)12$		X

** indicates a secondary essential prime implicant

$\circ$ indicates a distinguished row

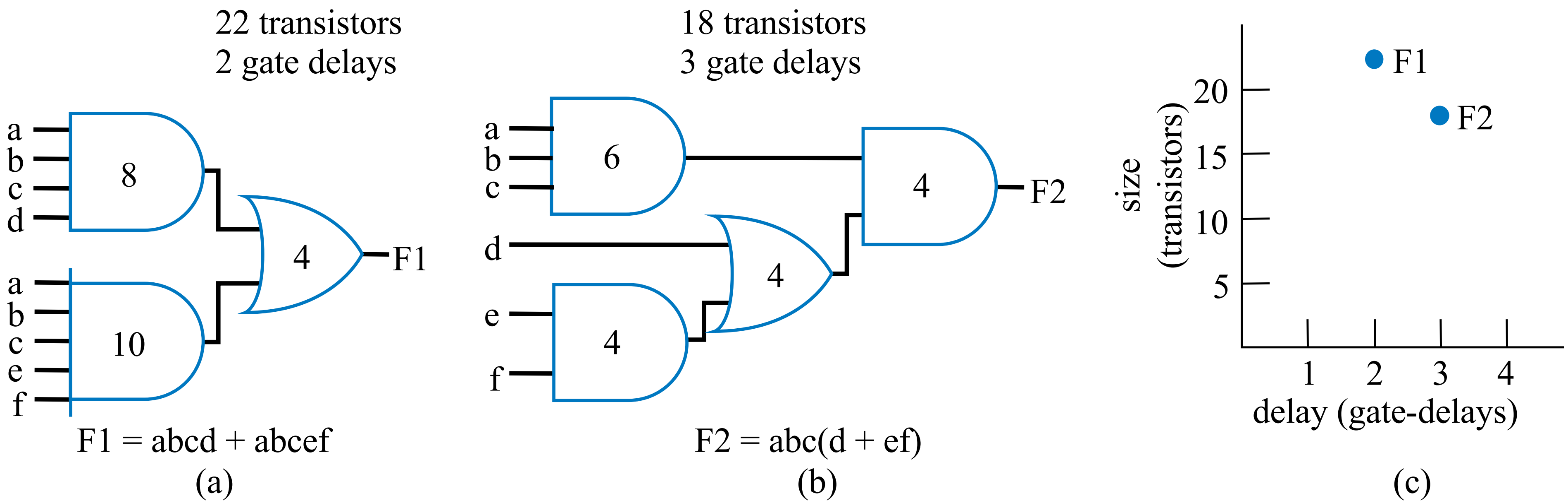
Multi-Level Logic Optimization – **Performance/Size Tradeoffs**

- We don't always need the speed of two-level logic
  - Multiple levels may yield fewer gates
  - Example
    - F1 = ab + acd + ace → F2 = ab + ac(d + e) = a(b + c(d + e))
    - General technique: Factor out literals – xy + xz = x(y+z)



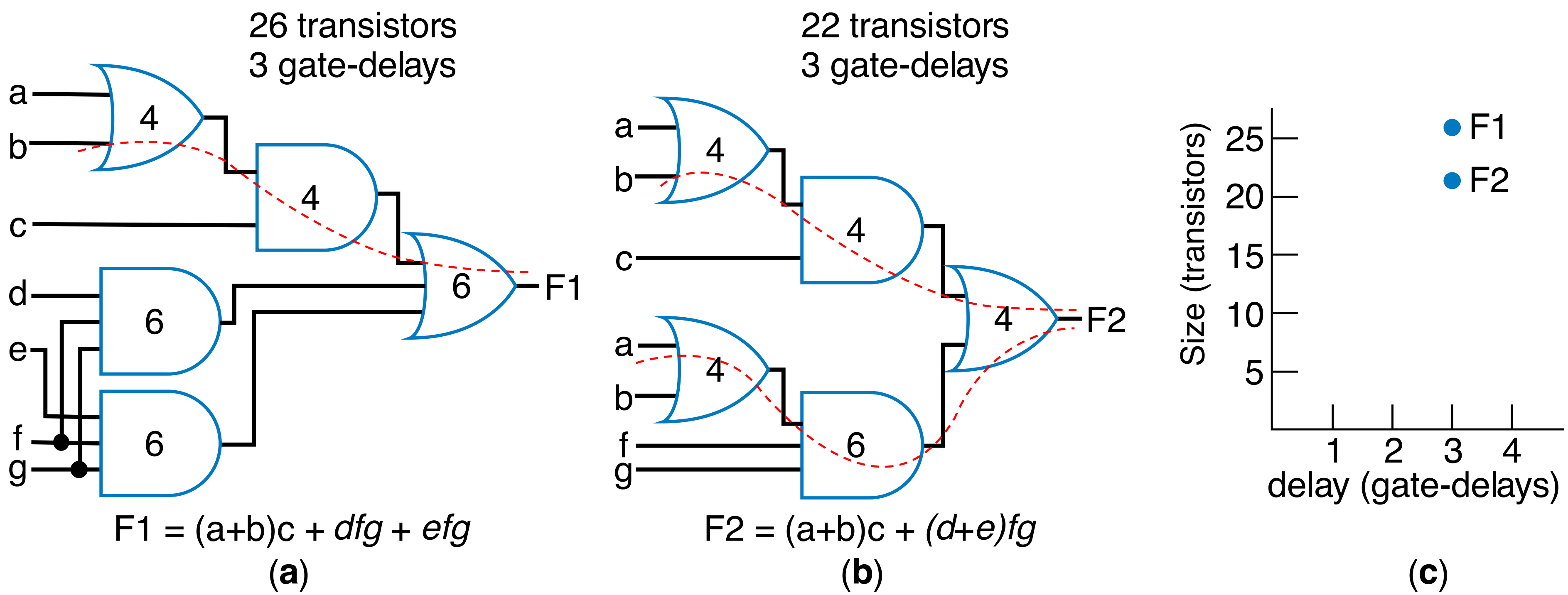
Multi-Level Example

- Q: Use multiple levels to reduce number of transistors for
  - F1 = abcd + abcef
- A: abcd + abcef = abc(d + ef)
  - Has fewer gate inputs, thus fewer transistors



Multi-Level Example: Non-Critical Path

- *Critical path*: longest delay path to output
- Optimization: reduce size of logic on non-critical paths by using multiple levels



Automated Multi-Level Methods

- Main techniques use heuristic iterative methods
  - Define various operations
    - "Factoring": abc + abd = ab(c+d)
    - Plus other transformations similar to two-level iterative improvement