

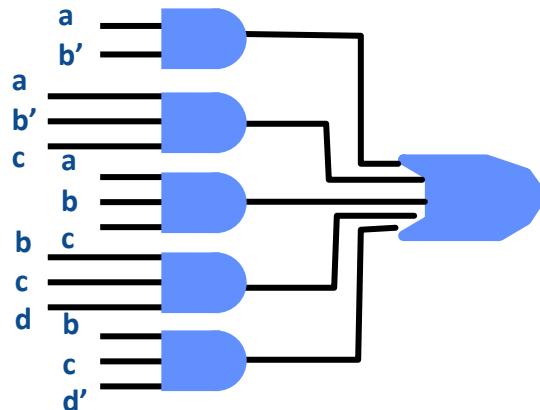
Lecture 5: Two-level Logic Synthesis



Two-level Synthesis Means ...

- Trying to find **minimal SOP logic**: many AND gates, 1 OR gate
 - Want: **fewest** AND gates, and among all such, one with **fewest input wires**.
 - These input wires called **literals**: 1 variable in true or complemented form in AND gate

$$f = ab' + ab'c + abc + bcd + bcd'$$



Good metric for success is to **minimize** the number of **literals** in this 2-level result

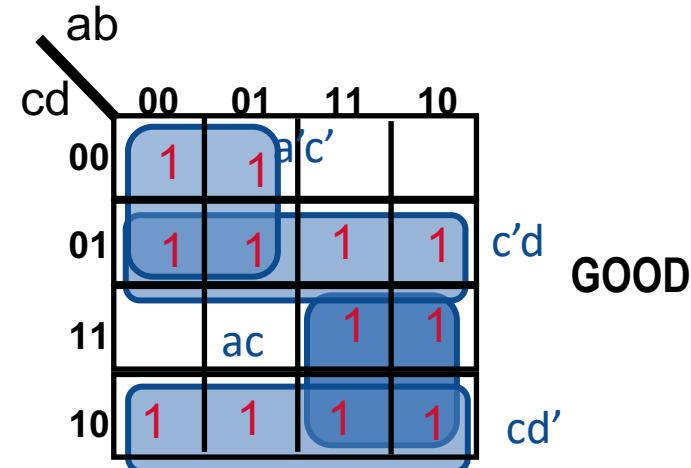
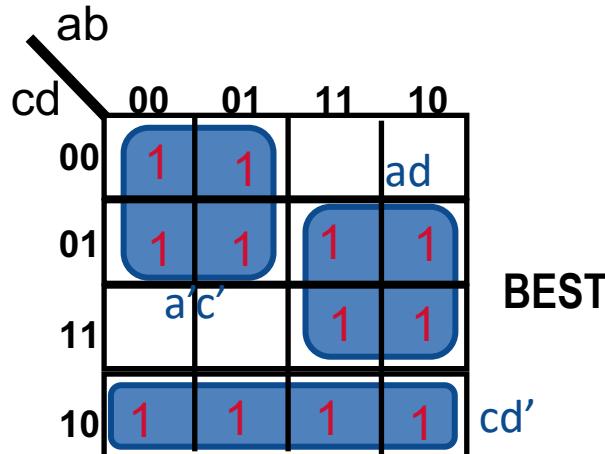
Two-level Minimization

- ❑ None of these methods you know is really effective, practical...
 - ❑ Boolean algebra: Hard with many variables. Can't tell when have a good solution
 - ❑ Kmaps: Same. Hard with many variables, can't tell when you're really done
 - ❑ Tabular solution: E.g, Quine McCluskey. Exponential complexity to get best result
- ❑ Need a better strategy
 - ❑ Big idea #1: **Don't** try for the best, perfect answer. Just get a **good** answer.
 - ❑ Big idea #2: **Iterative improvement.** From one answer, **reshape** the solution to discover a (possibly better) answer. Continue until no more improvement.

Best vs “Good Enough”

□ Comparing the two solutions

- Both are made of product terms (“**cubes**”) that are “as big as possible”. We insist on this. These products/cubes are called “**Prime Implicants**” (or “primes”).
- **Famous result from 1950s:** Best solution is composed of *cover of primes*



Cost vs All Solutions

- ❑ Neither solution can be improved by **removing** a prime
 - ❑ Both solutions are “**irredundant**”.
 - ❑ We also insist on this. Note: no simple path to get better. **Need different idea...**

	ab	cd	00	01	11	10
00			1	1		
01			1	1	1	1
11					1	1
10			1	1	1	1

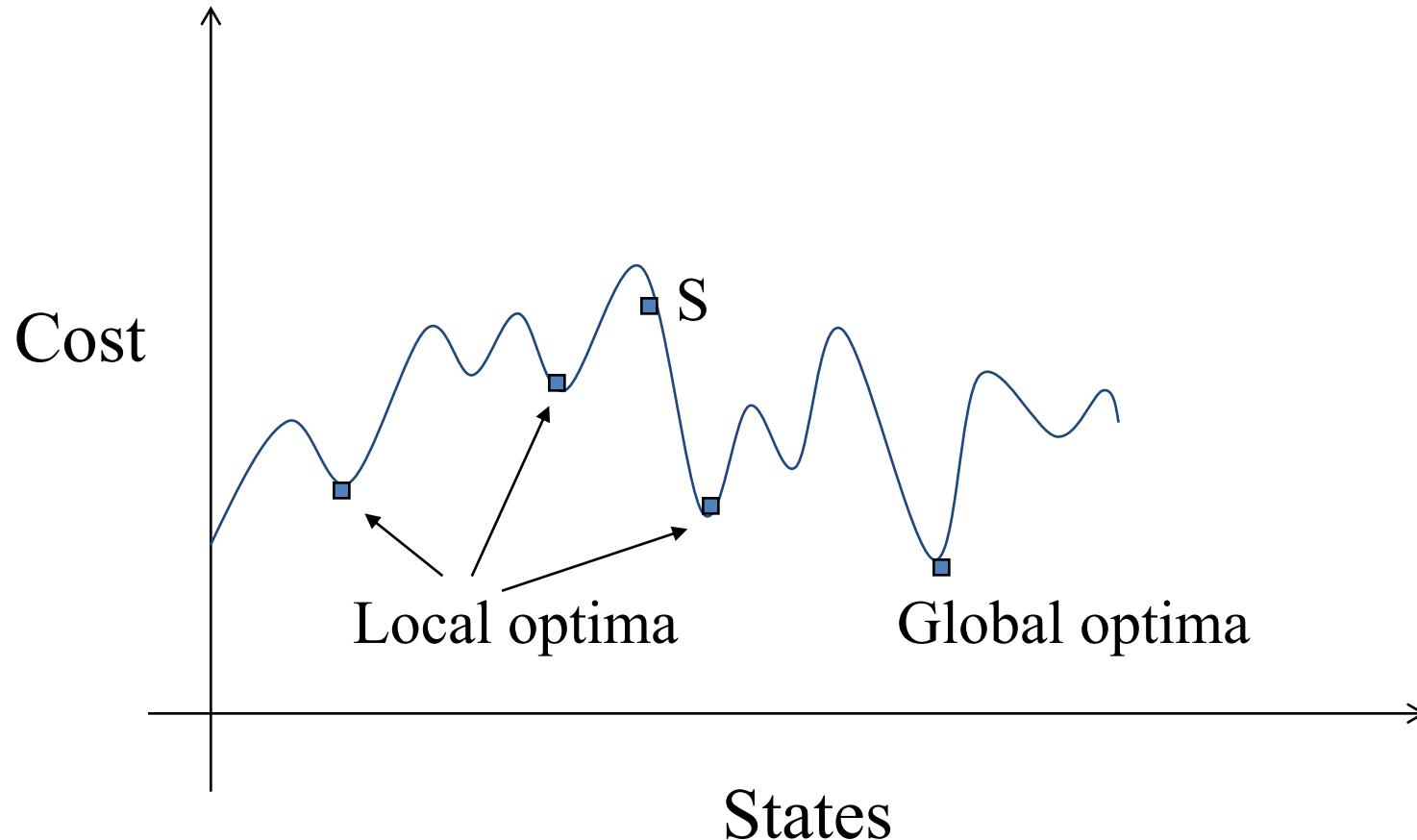
BEST

	ab	cd	00	01	11	10
00			1	1		
01			1	1	1	1
11					1	1
10			1	1	1	1

GOOD



Sounds Familiar?



Simulated Annealing Algorithm

Begin

Get an initial solution S and an initial temperature $T > 0$

while not yet “frozen” **do**

for $1 \leq i \leq P$ **do**

Pick a random neighbor S' of S ;

$\Delta = \text{Cost}(S') - \text{Cost}(S)$

if $\Delta \leq 0$ **then** $S \leftarrow S'$ // down-hill move

if $\Delta > 0$ **then** $S \leftarrow S'$ with probability $e^{-\Delta/T}$ // up-hill

$T \leftarrow rT$; // reduce temperature

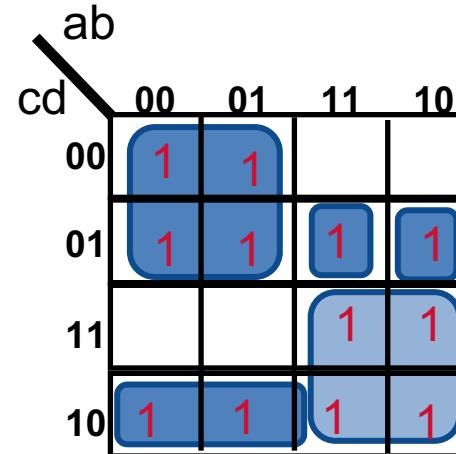
return S

End

The Reduce-Expand-Irredundant Optimization Loop

Assume we Start with a Truth Table

- But, might be a truth table (TT) with **input Don't Cares**
 - Just means each TT row can match **many** rows in a full truth table
 - Only list where the function is **1**; assume all other rows are **0**
 - Makes it easy to specify a function of many variables
 - Each row in this input TT defines a **product (cube)**— might **not** be prime, but it surely covers all the 1s

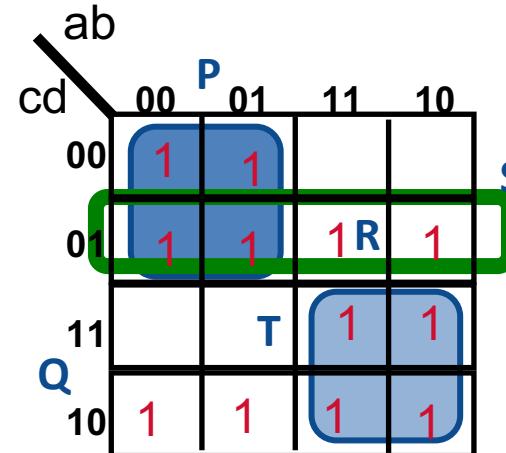


Our starting cover from this input TT

abcd	F	cube-label
0*0*	1	P
0*10	1	Q
1001	1	R
1101	1	S
1*1*	1	T

Next Step: Expand Each Cube to be Prime

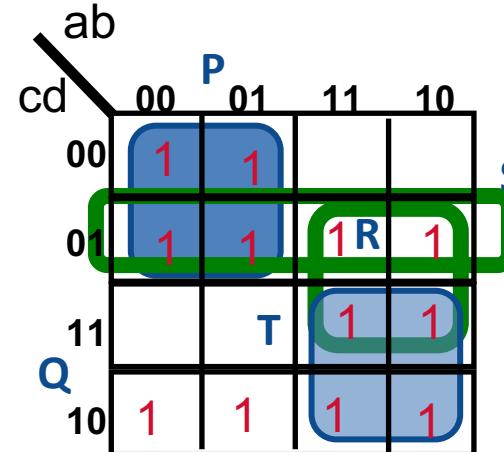
- “Expand” is a **heuristic**, done one cube at a time
 - Make each cube as **big as possible**
 - Might be *different* ways to do this for any specific cube...
 - **3** of our cubes have now been grown
 - Q, R, S cubes expanded
 - This new solution is a prime cover
 - But it might **not be best** we can do...



abcd	F	cube-label
0*0*	1	P
0*10	1	Q
1001	1	R
1101	1	S
1*1*	1	T

Next Step: Expand Each Cube to be Prime

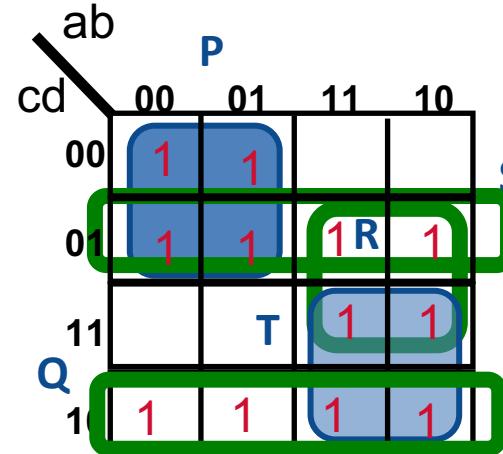
- “Expand” is a **heuristic**, done one cube at a time
 - Make each cube as **big as possible**
 - Might be *different* ways to do this for any specific cube...
 - **3** of our cubes have now been grown
 - Q, R, S cubes expanded
 - This new solution is a prime cover
 - But it might **not be best** we can do...



abcd	F	cube-label
0*0*	1	P
0*10	1	Q
1001	1	R
1101	1	S
1*1*	1	T

Next Step: Expand Each Cube to be Prime

- “Expand” is a **heuristic**, done one cube at a time
 - Make each cube as **big as possible**
 - Might be *different* ways to do this for any specific cube...
 - **3** of our cubes have now been grown
 - **Q, R, S** cubes expanded
 - This new solution is a prime cover
 - But it might **not be best** we can do...

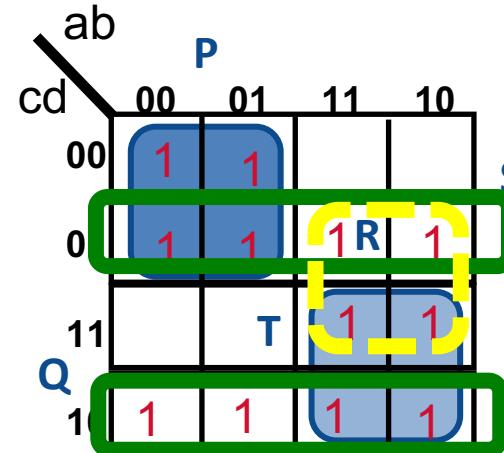


Expand cubes,
make them
prime

abcd	F	cube-label
0*0*	1	P
0*10	1	Q
1001	1	R
1101	1	S
1*1*	1	T

Next Step: Remove Irredundant Primes

- “Expand” is a **heuristic**, done one cube at a time
 - Make each cube as **big as possible**
 - Might be *different* ways to do this for any specific cube...
 - **3** of our cubes have now been grown
 - **Q, R, S** cubes expanded
 - This new solution is a prime cover
 - But it might **not be best** we can do...

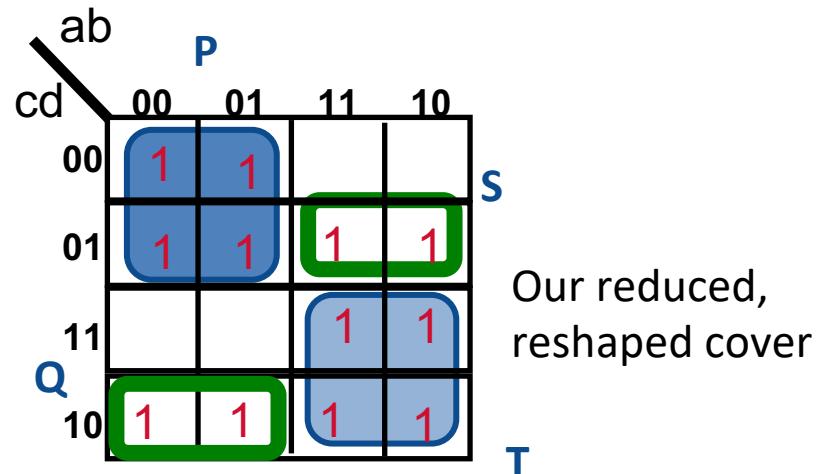


Expand cubes,
make them
prime

abcd	F	cube-label
0*0*	1	P
0*10	1	Q
1001	1	R
1101	1	S
1*1*	1	T

Next Step: Reduce the Prime Cover

- “Reduce” is another heuristic
 - Take each cube, “shrink it” as much as possible, but **do not uncover** any 1s.
 - These result cubes may **not** be prime; i.e. this is **not** necessarily a prime cover
- Surprising, *essential* step!
 - This new solution has different **shape**
 - **Big Idea:** When we expand it *again*, maybe we get a new, *better* solution
 - So, maybe we can still do *better*



abcd	F
0*0*	1
0*10	1
1001	1
1101	1
1*1*	1

Next Step: Expand Cubes Again

- Same “Expand” heuristic
 - But it is starting from a **different cover**, so can get a different answer!
 - Take each cube and **“expand”** it to make it prime, and also...
 - ...try to cover other cubes, to make them **redundant** (so we kill them later)
- In example: look at **T** cube!

ab	cd	00	01	11	10
cd	P	00	01	11	10
Q	00	1 1			S
01		1 1		1 1	
11			T	1 1	
10		1 1	1 1	1 1	

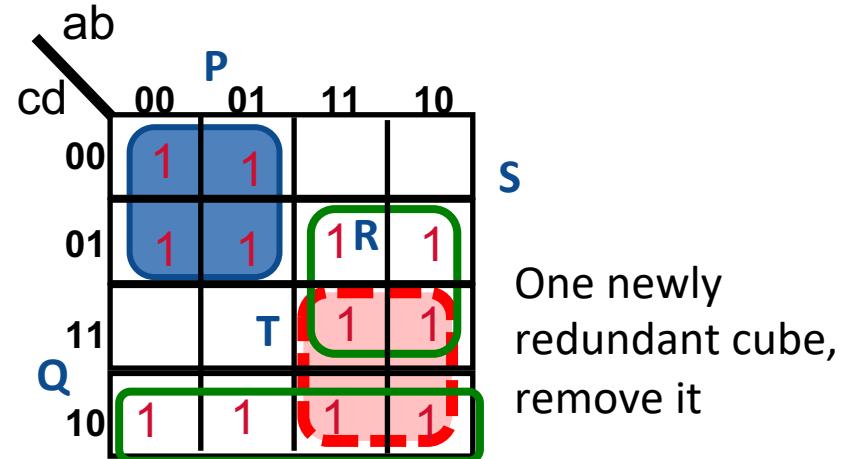
New, expanded cover, all cubes are prime again

abcd	F
0*0*	1
0*10	1
1001	1
1101	1
1*1*	1

Next Step: Check Redundant Again

- Same “Irredundant” heuristic
 - But it is starting from a **different cover**, so can get a different answer!
 - Took each cube and “**expanded**” it to make it prime, but we also...
 - ...tried to cover other cubes, to make them **redundant** (so we kill them later)

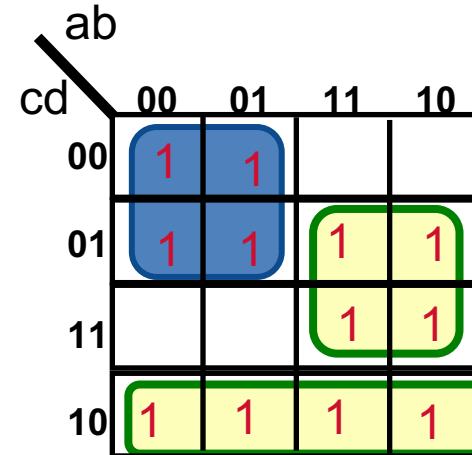
- In this example: we can kill another cube (**T**), it’s redundant
 - After this, the cover is again prime, and irredundant. Can’t remove anything to make it better (smaller)



abcd	F
0*0*	1
0*10	1
1001	1
1101	1
1*1*	1

This Result: Really Good!

- Got lucky: this is BEST answer
 - This will **not** generally happen
 - But we can guarantee a **prime, minimal, irredundant** solution
 - And it turns out in practice, that this iterative improvement “reshaping” of the cover produces excellent solutions

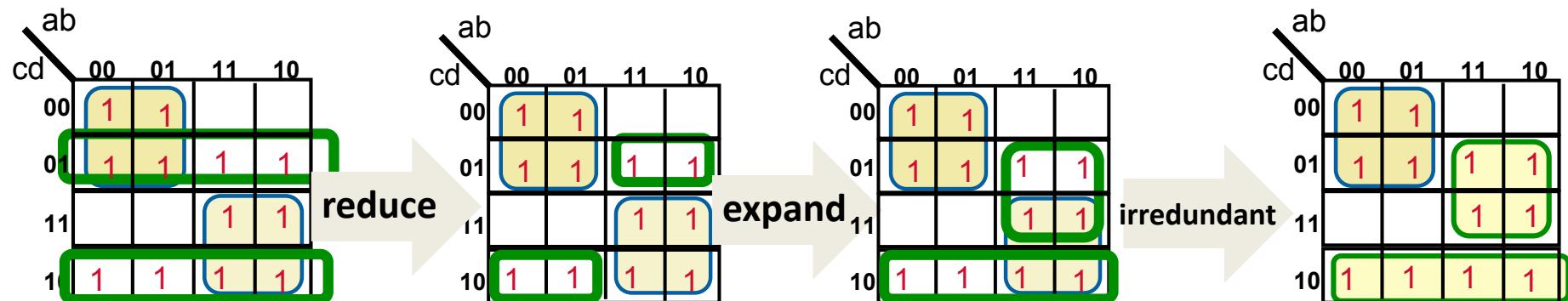


Final, optimized,
Prime, and
Irredundant Cover

abcd	F
0*0*	1
0*10	1
1001	1
1101	1
1*1*	1

Reduce-Expand-Irredundant Loop

- ❑ Famous tool: **ESPRESSO** for 2-level minimization
 - ❑ Started at IBM, finished at Berkeley
 - ❑ Brayton, Hachtel, McMullen, Sangiovanni-Vincentelli, **Logic Minimization Algorithms for VLSI Synthesis**, Kluwer Academic Press, 1984, is the reference here
 - ❑ Also, Giovanni DeMicheli, **Synthesis and Optimization of Digital Circuits**, McGraw Hill, 1994



ESPRESSO Algorithm Pseudocode

Input: F = ON-SET cover, D = DC-SET cover

$F = \text{Expand}(F, D);$

$F = \text{Irredundant}(F, D);$

repeat

| $\text{cost} = |F|;$

| $F = \text{Reduce}(F, D);$

| $F = \text{Expand}(F, D);$

| $F = \text{Irredundant}(F, D);$

until $|F| < \text{cost};$

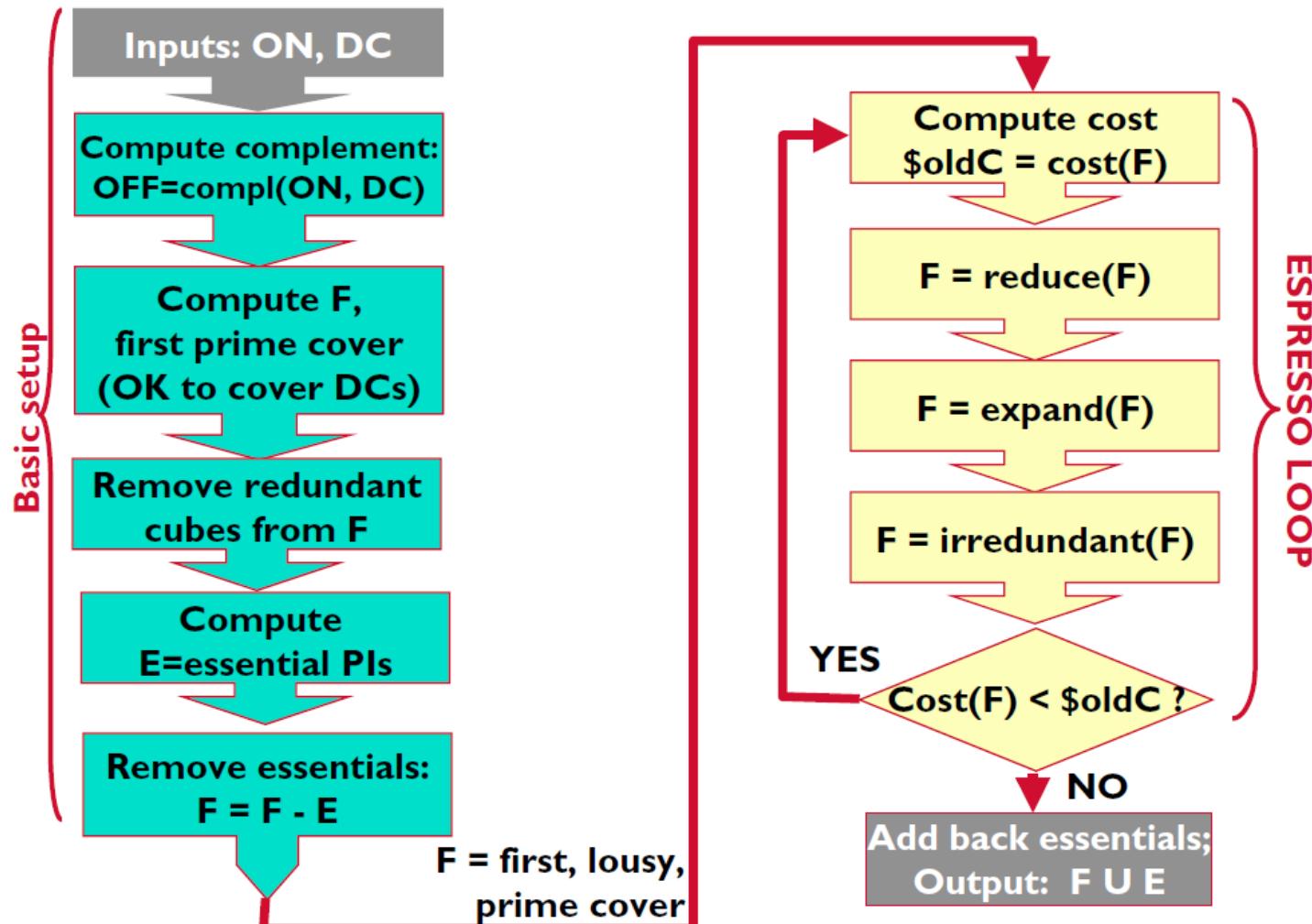
$F = \text{Make_Sparse}(F);$

ESPRESSO Algorithm

```
ESPRESSO (FON, FDC) {
    FOFF = complement(FON U FDC);
    F = expand(FON, FOFF);
    F = irredundant(FON, FDC)
    E = essentials(F, FDC);
    F = F - E;
    // get cover of OFF-set
    // get first cubelist cover of function f...
    // ...OK to cover some don't cares
    // get rid of redundant cubes from expand()
    // find essential primes, remember them
    // take essentials out of F, we don't need
    // to try to look later for covers of these

    // ESPRESSO loop
    do {
        $C = cost(cubelist for F);
        F = reduce(F, FDC);
        F = expand(F, FOFF);
        F = irredundant(F, FDC);
    } while( cost(cubelist for F) < $C)
    return( F U E );
}
```

ESPRESSO Flow



ESPRESSO Loop

I. How you do
complement,
why you need it.

```
ESPRESSO (FON, FDC) {  
    FOFF = complement(FON U FDC);  
    F = expand(FON, FOFF);
```

```
F = irredundant(FON, FDC)  
E = essentials(F, FDC);  
F = F - E;
```

Skip this one.

2. Simplified version
of how expand
works

```
do {  
    $C = cost(cubelist for F);  
    F = reduce(F, FDC);  
    F = expand(F, FOFF);  
    F = irredundant(F, FDC);  
} while( cost(cubelist for F) < $C)  
return( F U E );  
}
```

3. Just mention what
reduce does, not how.

4. Just mention what
irred. does, not how.

ESPRESSO: Collection of Elegant Heuristics

❑ Reduce-Expand-Irredundant cycle

❑ Reduce

- Rank cubes in a clever order, do PCN bit hacking to reduce them individually

❑ Expand

- Rank cubes in the opposite of this clever order, expand each individually as a pair of covering problems

❑ Irredundant

- A clever URP algorithm (like tautology) + a clever covering problem

❑ And a bunch of **other** interesting steps we did not mention...

Aside: Other Things Method Can Do

- ❑ Minimize **several** functions at the **same time**
 - ❑ Each function will be reduced to a 2-level form
 - ❑ But some product terms (AND gates) will be **shared**
 - ❑ This means: make this AND product once in hardware, connect it to many OR gate outputs to sum it into other functions. Can save a lot of hardware this way

- ❑ Handle conventional **Don't Cares**
 - ❑ Can specify a row of the truth table (TT) as being a “Don’t Care”
 - ❑ Means the hardware can make a **1** or a **0** as output for this input— you don’t care
 - ❑ Let algorithm choose **0** vs **1** output to make better, more minimal hardware.

How Well Does All This Work ... ?

- Fabulous: Very fast, very robust
- Where does ESPRESSO spend its time? [Brayton et al Kluwer book, 1984]
 - Complement 14% (big if there are lots of cubes in cover)
 - Expand 29% (depends on size of complement)
 - Irredundant 12%
 - Essentials 13% (some primes *must* be in answer; find them first)
 - Reduce 8%
 - Various optimizations 22% (special case optimizations)
- How fast?
 - Usually < 5 reduce-expand-irredundant iterations; often converges in just 1-2
 - Thousands of cubes, tens of thousands of literals: << 1 CPU second

Lets Look (Briefly) At One Step: *Expand*

- What does 'expand a cube' mean? **Remove** variables from cube

zw	xy	00	01	11	10
00		1		1	1
01		1		1	1
11				1	
10				1	

$$xyz'w' = [01 \ 01 \ 10 \ 10]$$

Remove
yw

zw	xy	00	01	11	10
00		1		1	1
01			1	1	1
11				1	
10				1	

$$xz' = [01 \ 11 \ 10 \ 11]$$

Remove
zw

zw	xy	00	01	11	10
00		1		1	1
01			1	1	1
11				1	
10				1	

$$xy = [01 \ 01 \ 11 \ 11]$$

Transform into a *Covering Problem*

□ Here is the most basic **Covering Problem**

- Given matrix of **R** rows x **C** columns. Matrix has 1s & 0s in it. Only draw the 1s.
- Choose **smallest set of rows** so that, using only these rows, every column has *at least* a single **1** in it – i.e., **every column is “covered” by the selected rows**
- Very good **heuristics** to get decent, fast solutions for these

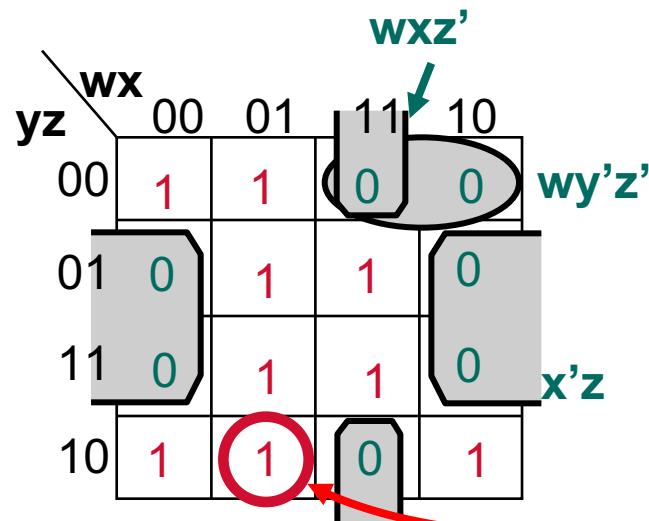
	C1	C2	C3	C4	C5
R1		1			
R2	1	1	1		
R3			1		1
R4		1		1	1

cover →

	C1	C2	C3	C4	C5
R1		1			
R2	1	1	1		
R3				1	
R4			1		1

The Blocking Matrix

- Expand = a Covering Problem on the Blocking Matrix
 - First: Given function F , build a **cube cover** of the **0s** in F
 - *called the OFF Set*
 - Why: We need to know what our cube **cannot touch** when it expands
 - How: **Complement** of the starting cover of the function! (Yes, this *exactly* our Programming Assignment #1, this is why we assigned it...)

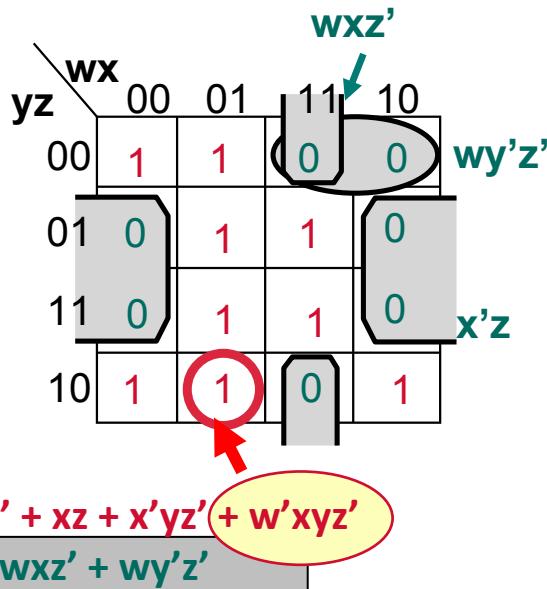


$$\begin{aligned} F &= w'y'z' + xz + x'yz' + w'xyz' \\ \text{Complement} &\quad \downarrow \\ F' &= x'z + wxz' + wy'z' \end{aligned}$$

Expand this cube

Build the *Blocking Matrix*

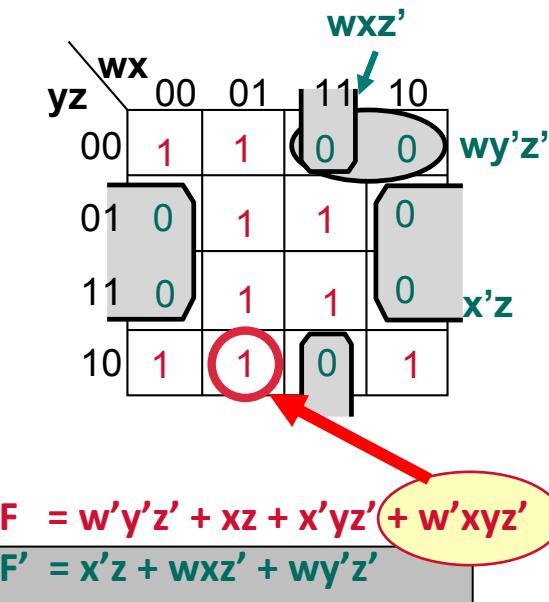
- This is a binary matrix structured as follows:
 - One **row** for each variable in the cube you are trying to expand
 - One **column** for each cube in the cover of the **OFF** set
 - Put a “**1**” in the matrix if the cube variable (row) **≠ polarity** of variable in the cube (column) of the **OFF** cover; else “**0**”. If don’t care, it’s a “**0**” (draw as *blank*)



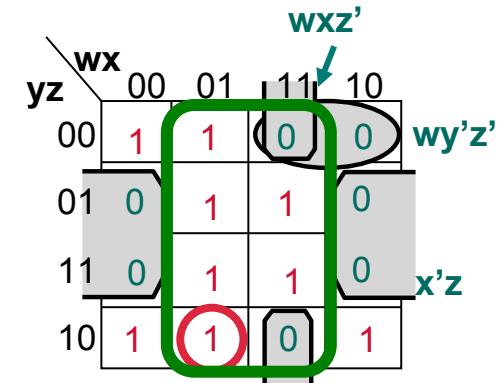
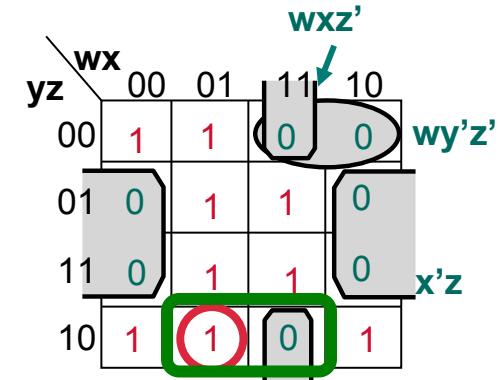
cubes in F' cover (may be a lot...)			
	$x'z$	wxz'	$wy'z'$
w'			
x			
y			
z'			

“1” in Blocking Matrix Means...

- Row: If you **remove** this variable (to expand cube)...
- Column: ...then you might **overlap** this OFF cube
 - Depending on what *other* variables you remove from this cube to expand it.
 - You cannot hit any **OFF** cubes.



	$x'z$	wxz'	$wy'z'$
w'		1	1
x	1		
y			1
z'	1		

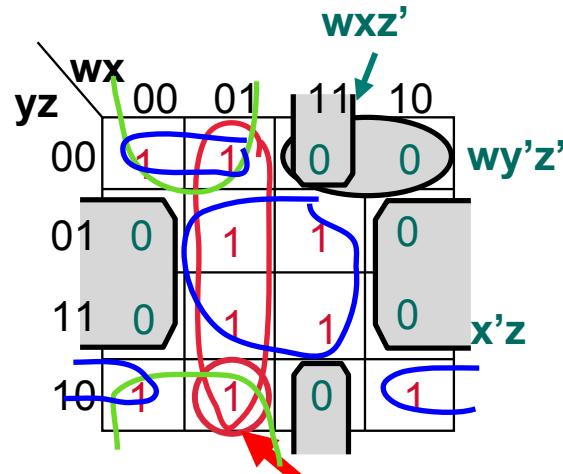


Minimum Cover in the Blocking Matrix

- Result: Find *smallest* set of rows that **covers** each column.

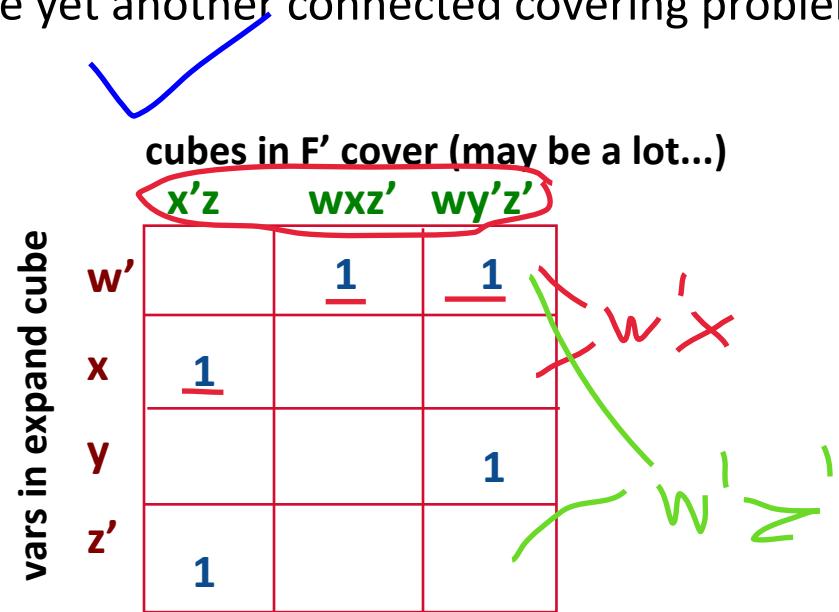
Product of these row variables is a **legal cube expansion**

- If you **keep** just these variables, you “mutually avoid” **ALL** the **OFF** cubes!
- (Also, try to **overlap** other cubes to make them **redundant**; not talking about this, but it turns out to be yet another connected covering problem...)



$$F = w'y'z' + xz + x'yz' + w'xyz'$$

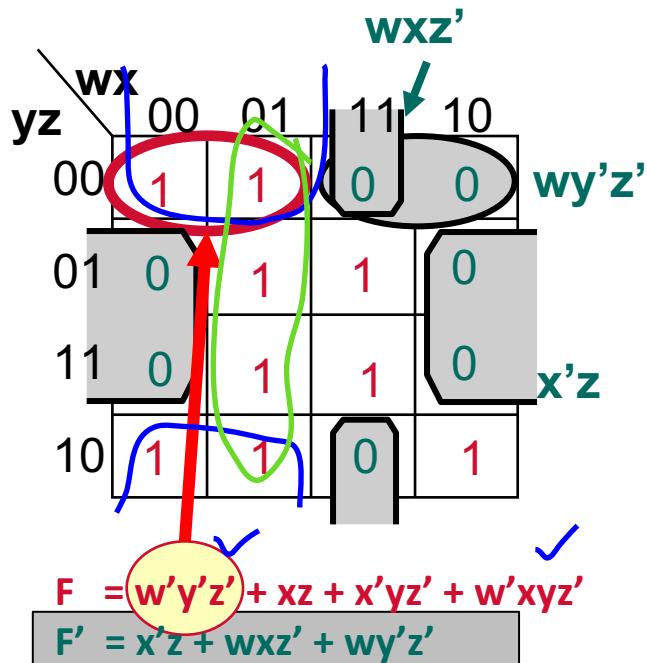
$$F' = x'z + wxz' + wy'z'$$



Let's walk through another example

- Let's try expanding $w'y'z'$

- Can we expand this?



vars in expand cube

cubes in F' cover

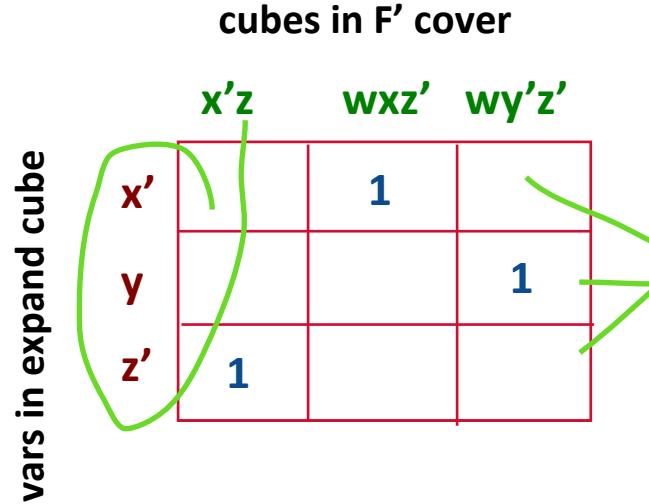
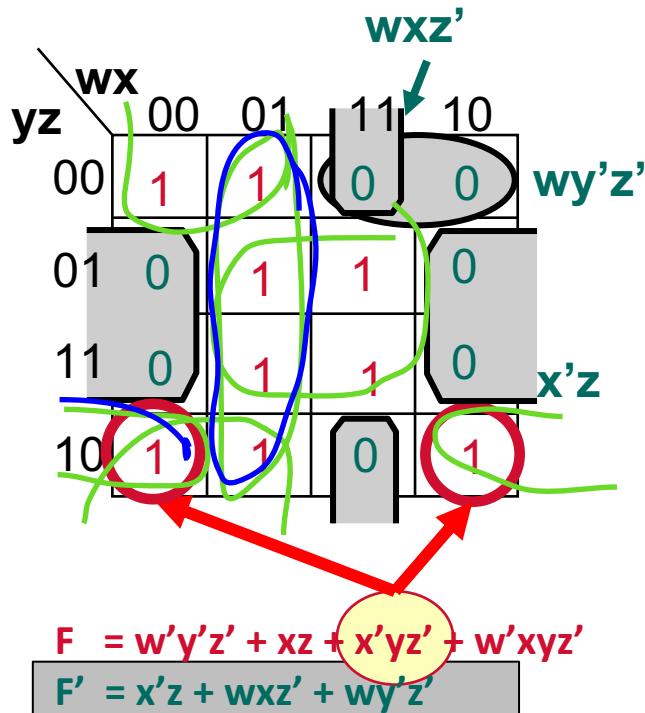
	$x'z$	wxz'	$wy'z'$
w'		1	1
y'			
z'	1		

W, I, I

Let's walk through another example

- Let's try expanding $x'yz'$

- Can we expand this?



ESPRESSO: Collection of Elegant Heuristics

- Reduce-Expand-Irredundant cycle
 - Reduce
 - Rank cubes in a clever order
 - Take a step back to start from a “worse” solution
 - Expand
 - Rank cubes in the opposite of this clever order, expand each individually as a pair of covering problems
 - Irredundant
 - A clever URP algorithm (like tautology) + a clever covering problem
- And a bunch of **other** interesting steps we did not mention...

How Well Does All This Work ... ?

- ❑ Fabulous: Very fast, very robust
- ❑ Where does ESPRESSO spend its time? [Brayton et al Kluwer book, 1984]
 - ❑ Complement 14% (big if there are lots of cubes in cover)
 - ❑ Expand 29% (depends on size of complement)
 - ❑ Irredundant 12%
 - ❑ Essentials 13% (some primes *must* be in answer; find them first)
 - ❑ Reduce 8%
 - ❑ Various optimizations 22% (special case optimizations)
- ❑ How fast?
 - ❑ Usually < 5 reduce-expand-irredundant iterations; often converges in just 1-2
 - ❑ Thousands of cubes, tens of thousands of literals: << 1 CPU second

Summary

- ❑ 2-level logic synthesis uses **heuristics** to find good solutions
 - ❑ Not “best”, but instead “good enough”
 - ❑ Minimal (not minimum), prime, irredundant
 - ❑ Famous idea: iterative improvement – **reduce-expand-irredundant loop**
 - ❑ All done with PCN cubelists, covering matrices, and URP ideas
- ❑ Not every piece of logic is implemented in 2-level form ...
- ❑ Next: **Multi-level logic**