Announcements

- Lab3 is due on Friday (9/19/2025) at 11:59PM.
  - This is the first Hardware simulation Lab
- HW3 is due tonight (9/17/2025)
  - Upload your completed HW3 into Gradescope
- MIDTERM1 is on 9/23/2025 (7:15-9:15PM)
  - Exam's Detailed Instructions/Guidelines/Protocols along with practice tests are posted on Canvas Exam Schedule page.
  - **Conflict request deadline is today by 5:00PM**

University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Boolean Expression Terminology

Sum-of-Products (SOP) Form is Quite Common

sum-of-products (SOP)

a sum (OR)
of products (AND)
of literals

examples: **AB + BC**,

AB' + C + A'C'D',

but NOT **A'(B + C) + D**

Product-of-Sums (POS) Form is Also Common

product-of-sums (POS)

a product (AND)
of sums (OR)
of literals

examples: $(A + B)(B + C)$,

$(A + B')C(A' + C' + D')$,

but NOT $(A + BC)D$

Canonical Forms Allow Easy Comparison, But Are Too Big

canonical SOP

a sum of minterms; the expression produced by the logical completeness construction

canonical POS

a product of maxterms

What does canonical mean?

Canonical Forms Allow Easy Comparison, But Are Too Big

canonical SOP

a sum of minterms; the expression produced by the logical completeness construction

canonical POS

a product of maxterms

What does canonical mean?

Unique (if we assume an ordering on variables).

Canonical Forms Allow Easy Comparison, But Are Too Big

canonical SOP

a sum of minterms; the expression produced by the logical completeness construction

canonical POS

a product of maxterms

What does canonical mean?

Unique (if we assume an ordering on variables).

Too many terms to be of practical value.

Do You Know Mathematical Implication?

What does A→B mean?

Do You Know Mathematical Implication?

What does A→B mean?

A implies B.

Do You Know Mathematical Implication?

What does A→B mean?

A implies B.

In other words: **if A is true, B is also true.**

Do You Know Mathematical Implication?

What does A→B mean?

A implies B.

In other words: **if A is true, B is also true.**

What if **A is false?**

Do You Know Mathematical Implication?

What does A→B mean?

A implies B.

In other words: **if A is true, B is also true.**

What if **A is false?**

In that case, **is A→B true or false?**

Do You Know Mathematical Implication?

What does A→B mean?

A implies B.

In other words: **if A is true, B is also true.**

What if **A is false?**

In that case, **is A→B true or false?**

If A is false, A→B is true.

So the Following Odd Statements are True

All **purple elephants** can fly.

(X is a **purple elephant** → X can fly.)

Students who score **above 125%** in ECE120 fail the class.

(X scored **above 125%** → X fails.)

In both, **the premise is false for any X**, so the **implications are true**.

One Function Can Imply Another

A function **G is an implicant of** a second function **F iff G** operates on the same variables as **F** and **G→F**.

In other words, every row

- with an output of 1 in **G**'s truth table
- also has an output of 1 in **F**'s truth table.

0 rows in **G**'s truth table do not matter.

For Our Purposes, Implicants are Products of Literals

In digital design, we only refer to **products of literals as implicants**.

So we will **assume that an implicant can be written as a product of literals**.

For Our Purposes, Implicants are Products of Literals

In digital design, we only refer to
products of literals as implicants.

For example, take **F = AB'C + ABC' + ABC**.

Each product term is an implicant of F.

(e.g. when **AB'C is 1 F is also 1)**

We Can Use Implicants to Simplify Functions

As a first step towards simplifying a function **F**, we can ask:

Given an implicant G of F, can we remove any of its literals and obtain another implicant of F?

For example, take **F = AB'C + ABC' + ABC**.

The first term (**AB'C**) is an implicant.

Can we remove any literals?

Try to Remove Each Literal to Find Only AC Implies F

Start from **AB'C** and try to remove each literal.

A	B	C	F	B'C	AC	AB'
0	0	0	0	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	1
1	0	1	1	1	1	1
1	1	0	1	0	0	0
1	1	1	1	0	1	0

Try to Remove Each Literal to Find Only AC Implies F

Start from **AB'C** and try to remove each literal.

A	B	C	F	B'C	AC	AB'
0	0	0	0	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	1
1	0	1	1	1	1	1
1	1	0	1	0	0	0
1	1	1	1	0	1	0

Try to Remove Each Literal to Find Only AC Implies F

Start from **AB'C** and try to remove each literal.

B'C is not an implicant.

A	B	C	F	B'C	AC	AB'
0	0	0	0	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	1
1	0	1	1	1	1	1
1	1	0	1	0	0	0
1	1	1	1	0	1	0

Try to Remove Each Literal to Find Only AC Implies F

Start from **AB'C** and try to remove each literal.

B'C is not an implicant.

AB' is not an implicant.

AC is an implicant.

A	B	C	F	B'C	AC	AB'
0	0	0	0	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	1
1	0	1	1	1	1	1
1	1	0	1	0	0	0
1	1	1	1	0	1	0

We Remove as Many Literals as We Can

So we can simplify **F** by replacing **AB'C** with **AC**:

$$F = AC + ABC' + ABC$$

Checking the second term (**ABC'**), we find that we can eliminate **C'** to obtain:

$$F = AC + AB + ABC$$

In the third term (**ABC**), we can eliminate **B** or **C**, but not both. Let's pick **B**.

$$F = AC + AB + AC$$

Prime Implicants Have a Minimal Number of Literals

$$\mathbf{F = AC + AB + AC}$$

But now we have a duplicate term, which we can eliminate to arrive at a simple form for **F**:

$$\mathbf{F = AC + AB}$$

We can remove no more literals.

One more definition: An implicant **G** of **F** is a **prime implicant of F** iff **none of the literals in G can be removed** to produce other implicants of **F**.

AB and AC are prime implicants of F.

To Simplify, Write Function as a Sum of Prime Implicants

The conclusion is obvious:

To simplify a function F,
write it as a sum of prime implicants.

To Simplify, Write Function as a Sum of Prime Implicants

The conclusion is obvious:

To simplify a function F,
write it as a sum of prime implicants.

Enjoy the algebra.

To Simplify, Write Function as a Sum of Prime Implicants

The conclusion is obvious:

**To simplify a function F,
write it as a sum of prime implicants.**

Enjoy the algebra.

Good luck!

(Next, we'll develop a graphical tool
that lets us skip the algebra.)

University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

Karnaugh Maps (K-Maps)

To Simplify, Write Function as a Sum of Prime Implicants

One way to simplify a function F:

Choose a set of prime implicants that, when ORed together, give F.

But our approach for picking prime implicants is not so easy.

The Domain of a Boolean Function is a Hypercube

We can

- **represent the domain**
- of a Boolean function **F** on **N** variables
- **as an N-dimensional hypercube**.

Each vertex in the hypercube corresponds to one combination of the **N** inputs.

The function **F** thus **has one value for each vertex** (each input combination).

List All Implicants for Two Variables, A and B

Now consider two input variables, **A** and **B**.

How many implicants are possible?

Start with minterms…

AB AB′ A′B A′B′

And products of one literal…

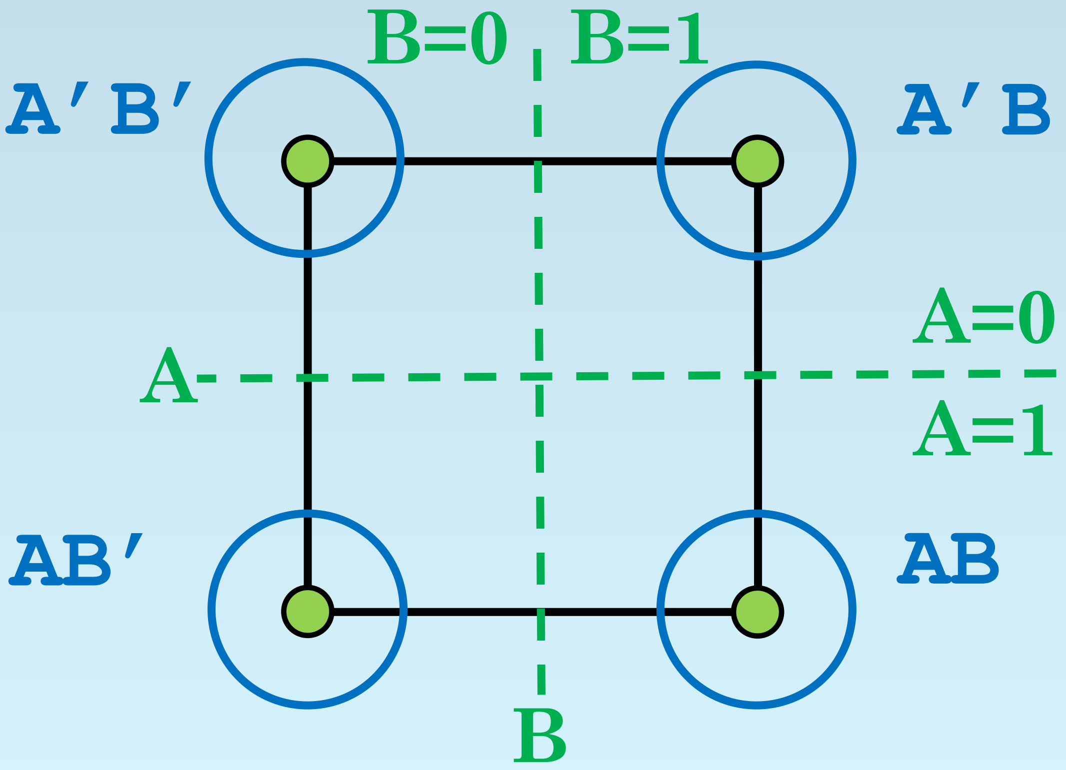
A A′ B B′

And, of course …

1

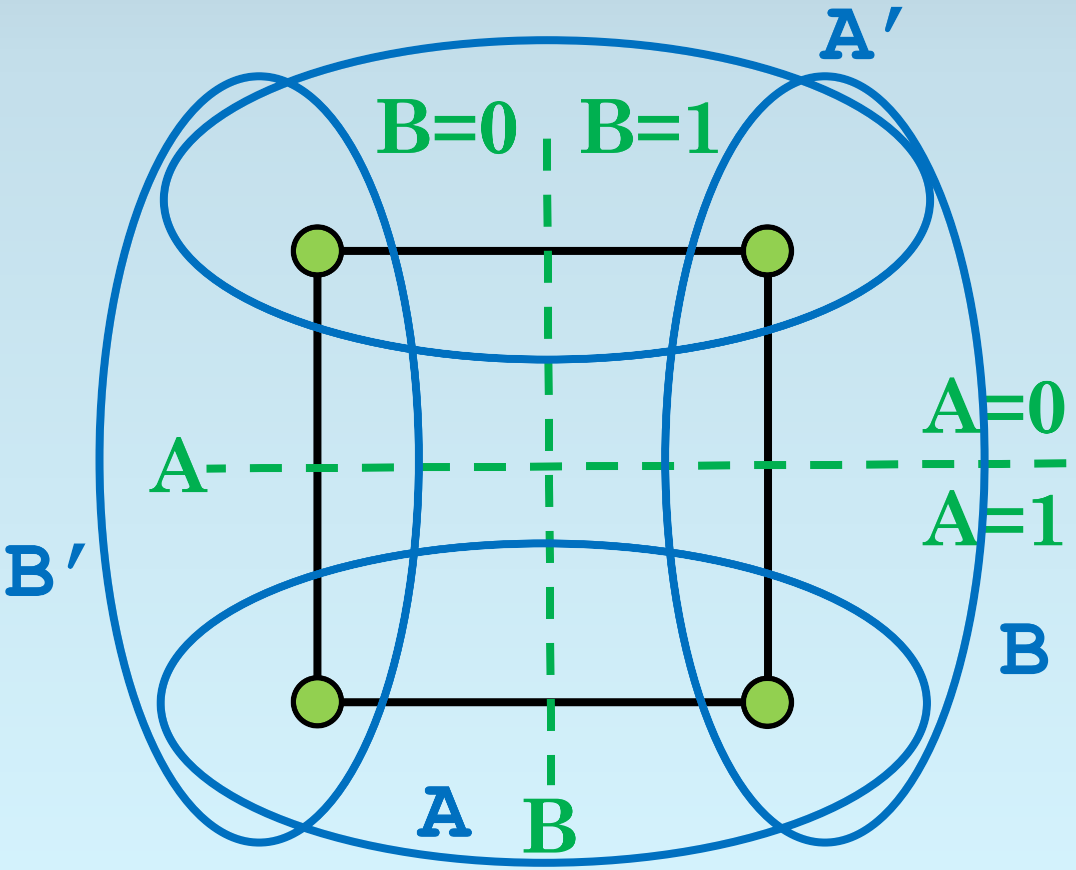
Minterms Correspond to Vertices

With **N = 2** (inputs A and B), a hypercube is a square: four vertices, four edges, and a face.



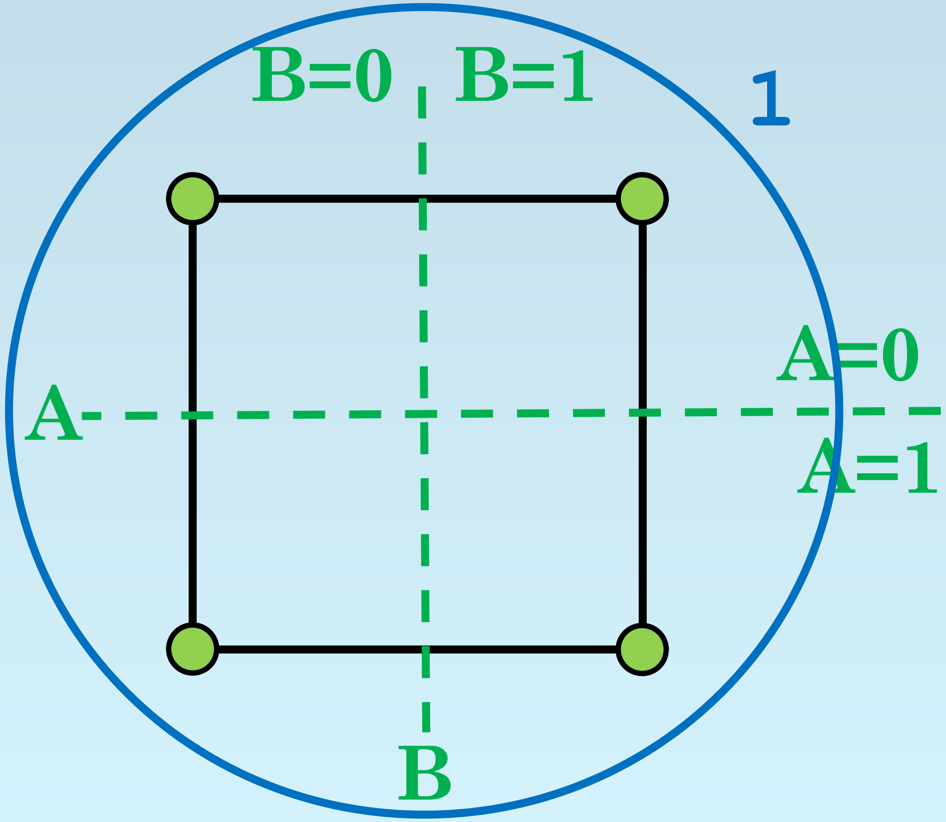
Single-Literal Implicants Correspond to Edges

Edges include both values of one variable.



The Implicant 1 Corresponds to the Face/Square

The face includes both values of both variables.



The Domain of a Boolean Function is a Hypercube

We can

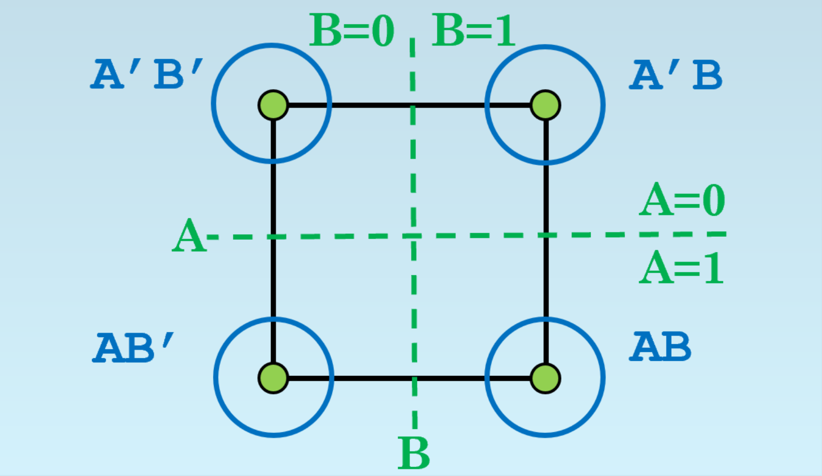
- **represent the domain**
- of a Boolean function **F** on **2** variables
- **as an 2-dimensional hypercube**.

Each vertex in the hypercube corresponds to one combination of the **2** inputs.

The function **F** thus **has one value for each vertex** (each input combination).

Let's consider a G(A,B) that we want to simplify

A	B	G(A,B)
0	0	1
0	1	1
1	0	0
1	1	1



G = A'B' + A'B + AB

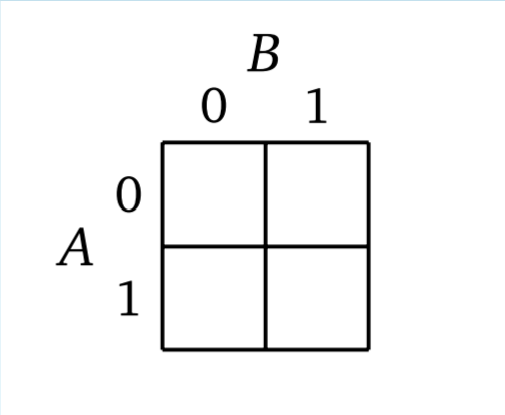
Can we simplify the G?

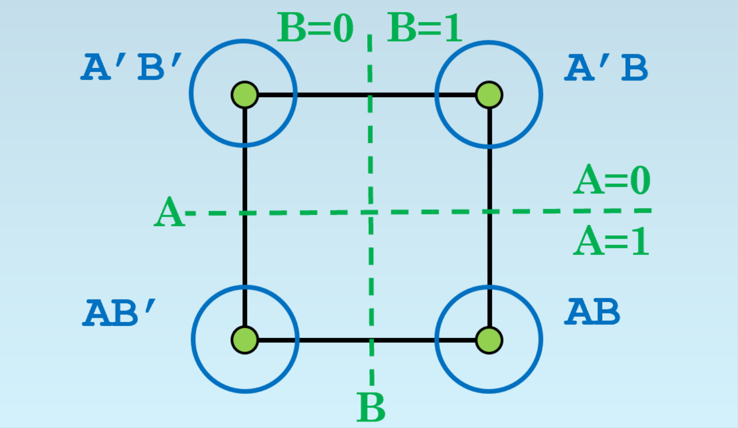
We Draw Function G(A,B) Using a 2-Variable K-Map

We can draw a **K-map** on 2 variables for the function **G(A,B)** as shown below.

Each box represents

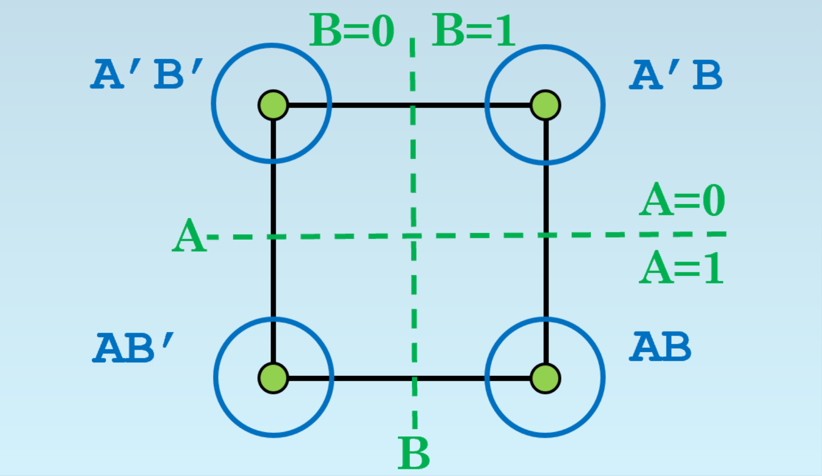
- an input combination
- a vertex of the hypercube, and
- **an implicant (a minterm)**.

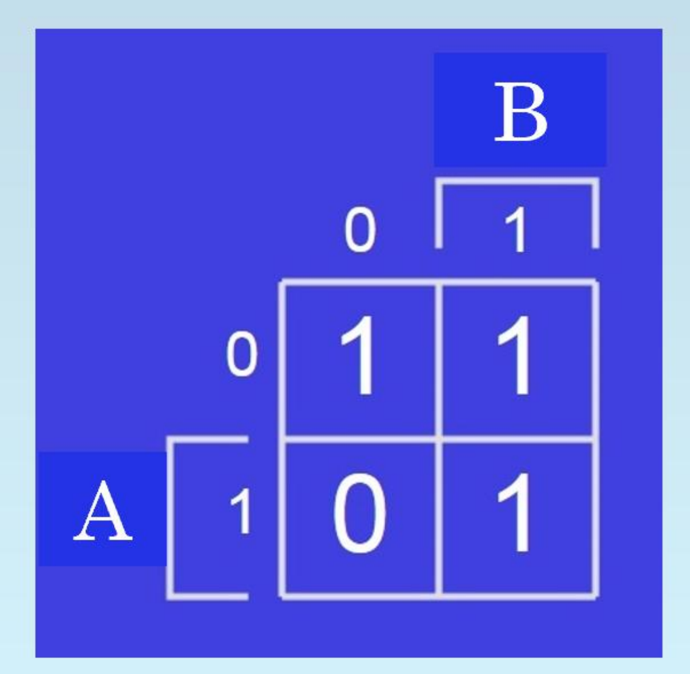




Let's consider a G(A,B) that we want to simplify

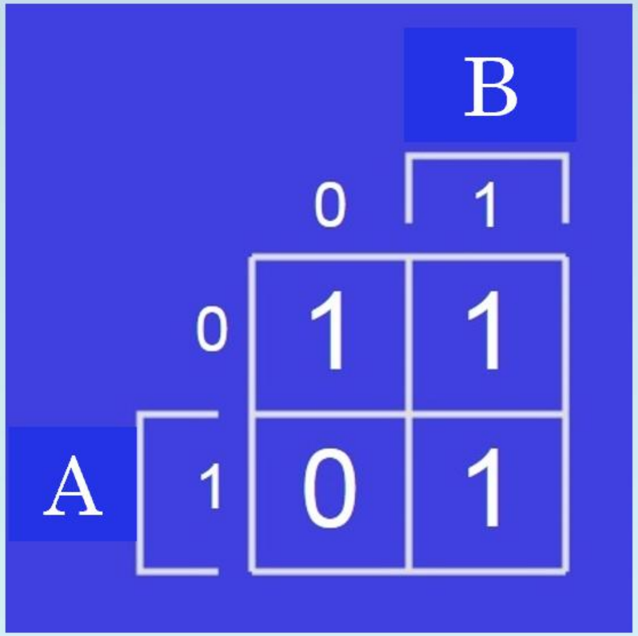
A	B	G(A,B)
0	0	1
0	1	1
1	0	0
1	1	1

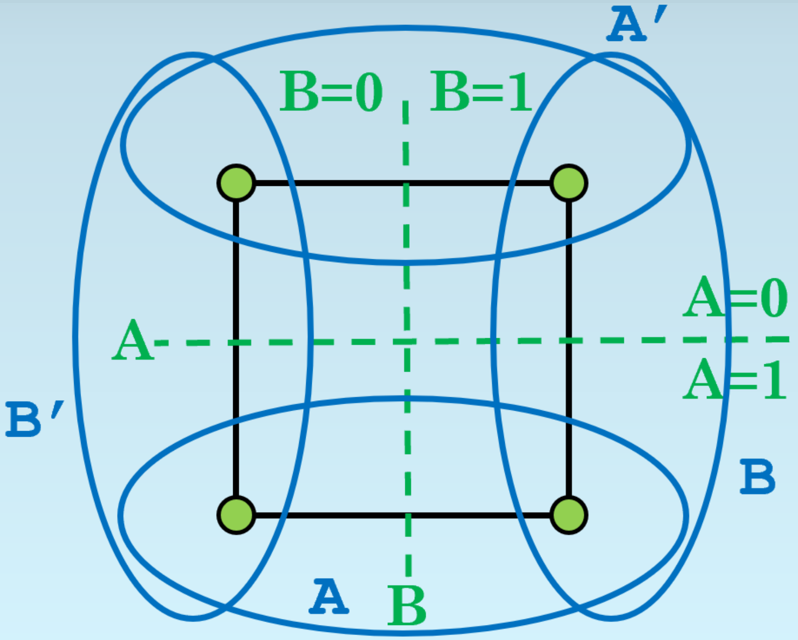




Let’s consider a G(A,B) that we want to simplify

A	B	G(A,B)
0	0	1
0	1	1
1	0	0
1	1	1





G = A’B’ + A’B + AB

Can we simplify the G?

Again, Grow the Loop Until We Get a Prime Implicant

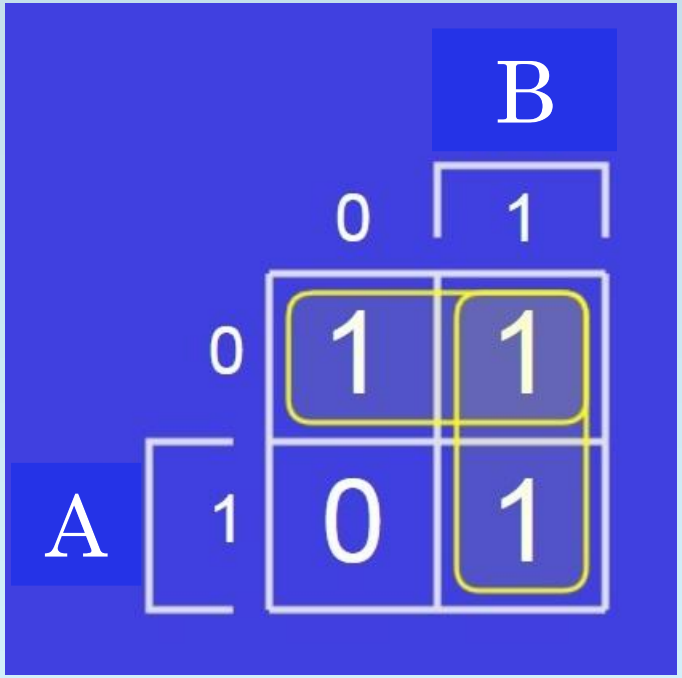
Let's grow the loop.

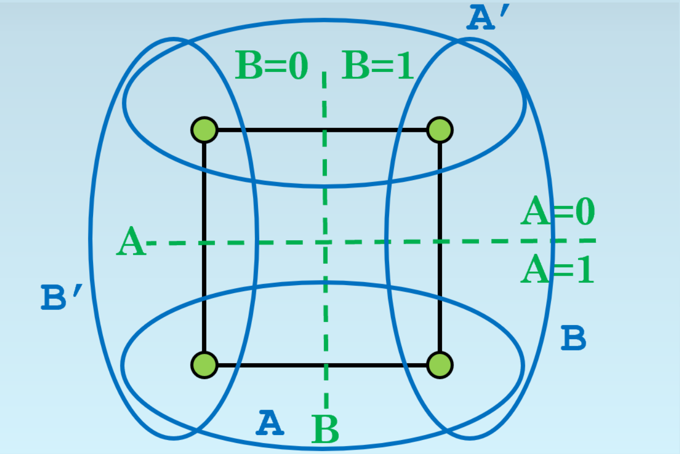
Together, these two loops cover all 1s in **G(A,B)**.

So we can write

G(A,B) = A' + B

<u>Now</u> are you excited?





List All Implicants for Variables A, B, and C

Guess what's next.

Three input variables: **A**, **B**, and **C**!

How many implicants are possible?

That's right: lots.

A 3D hypercube is a cube.

Let's count features instead.

A 3D Hypercube Has Vertices, Edges, Faces, and Cube

Now, let's count.

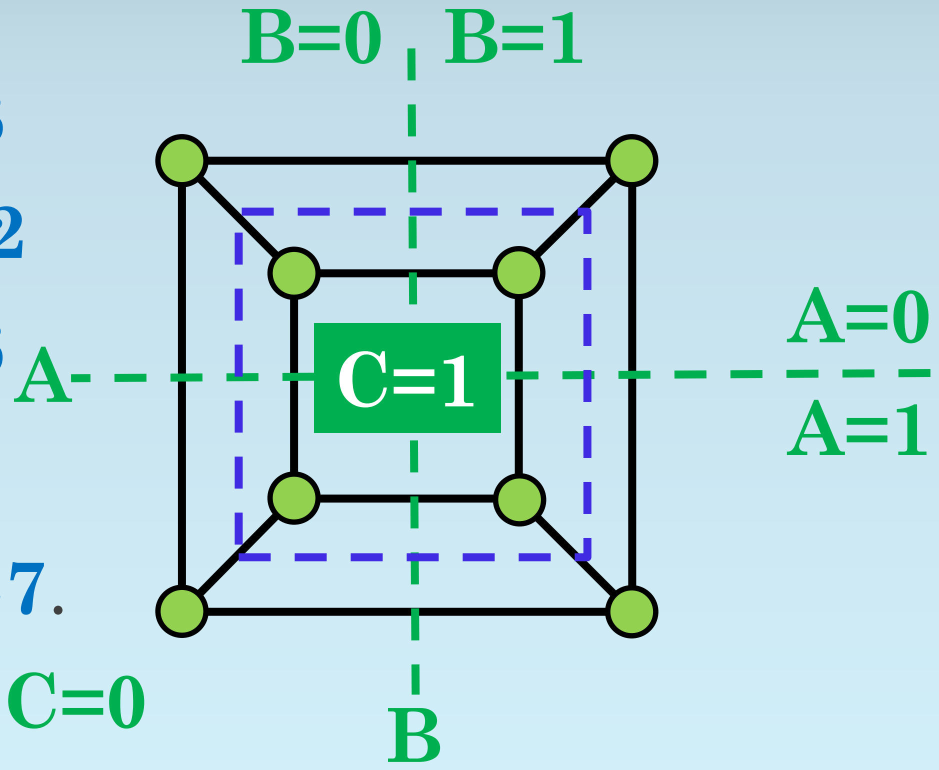
# of vertices? 8

# of edges? 12

# of faces? 6

# of cubes? 1

So the **total is 27**.

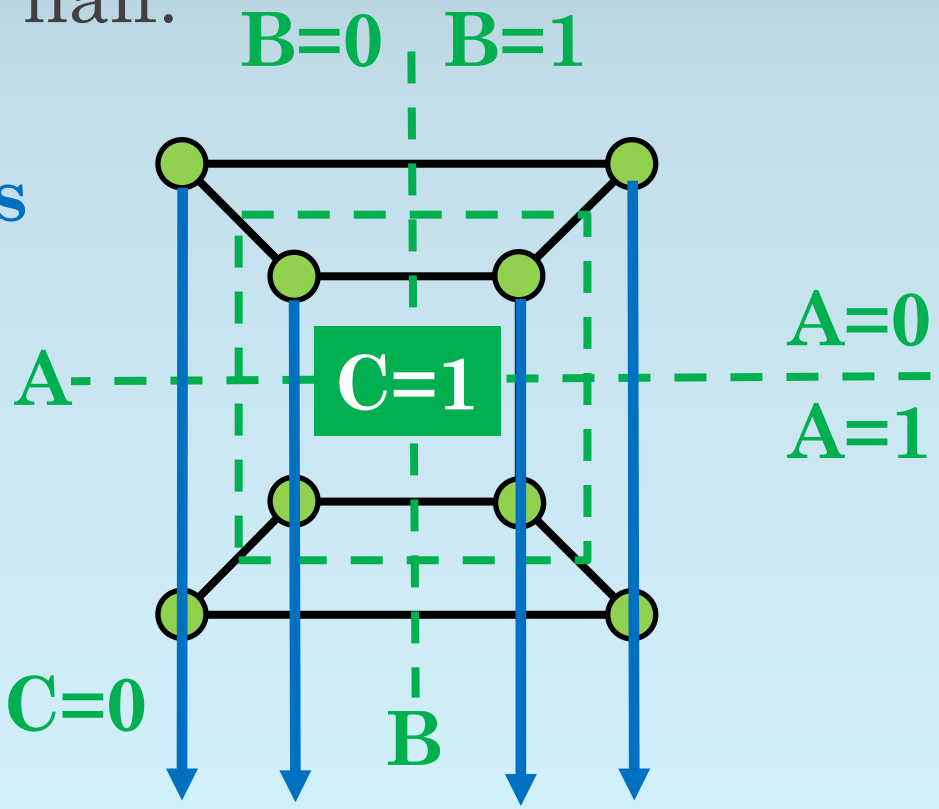


How Can We Draw Boxes for the Cube?

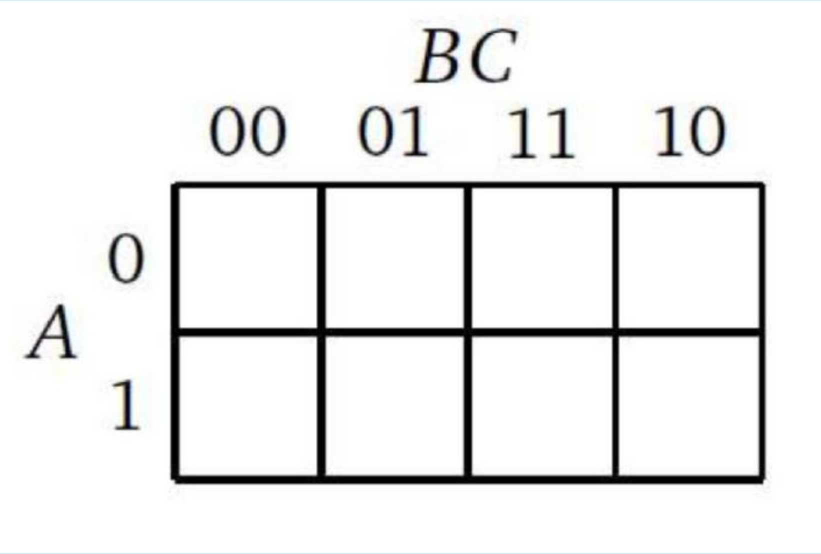
Focus on the top half.

Each adjacent B,C pair shares an edge.

The last edge wraps around (from 10 to 00).

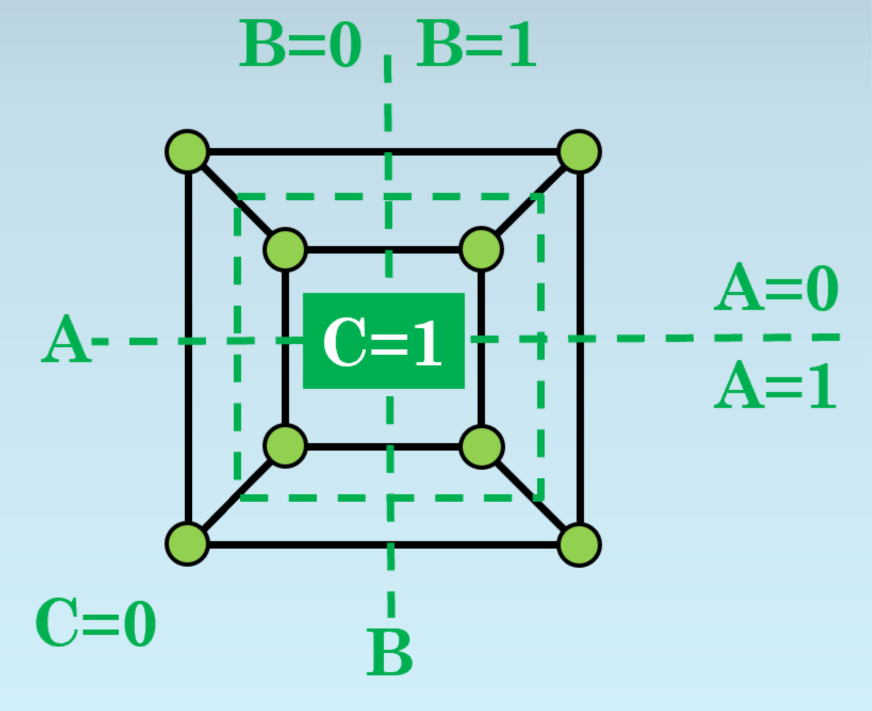


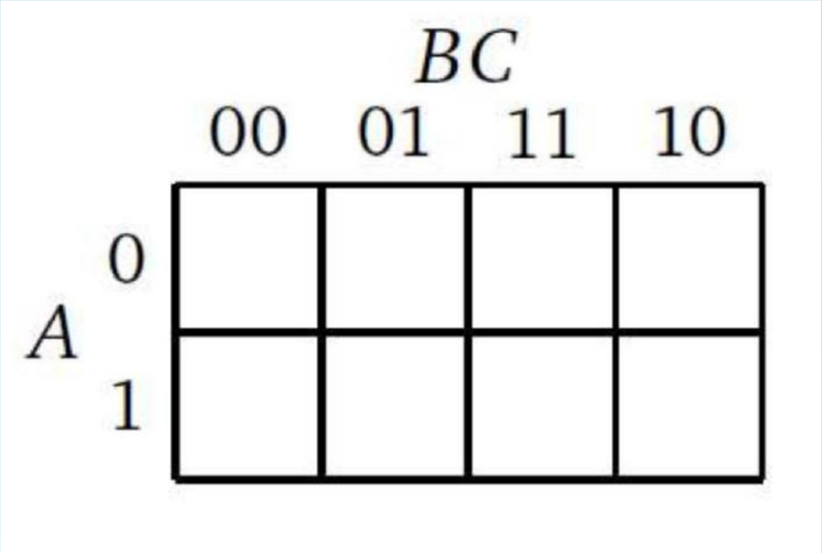
value of B,C 00 01 11 10



Gray code order

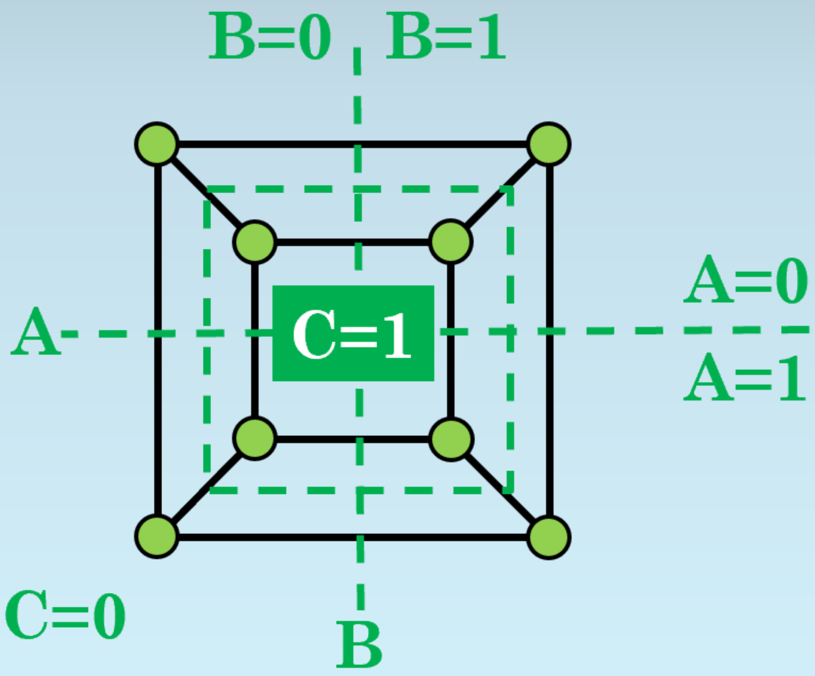
Let's consider a H(A,B,C) that we want to simplify

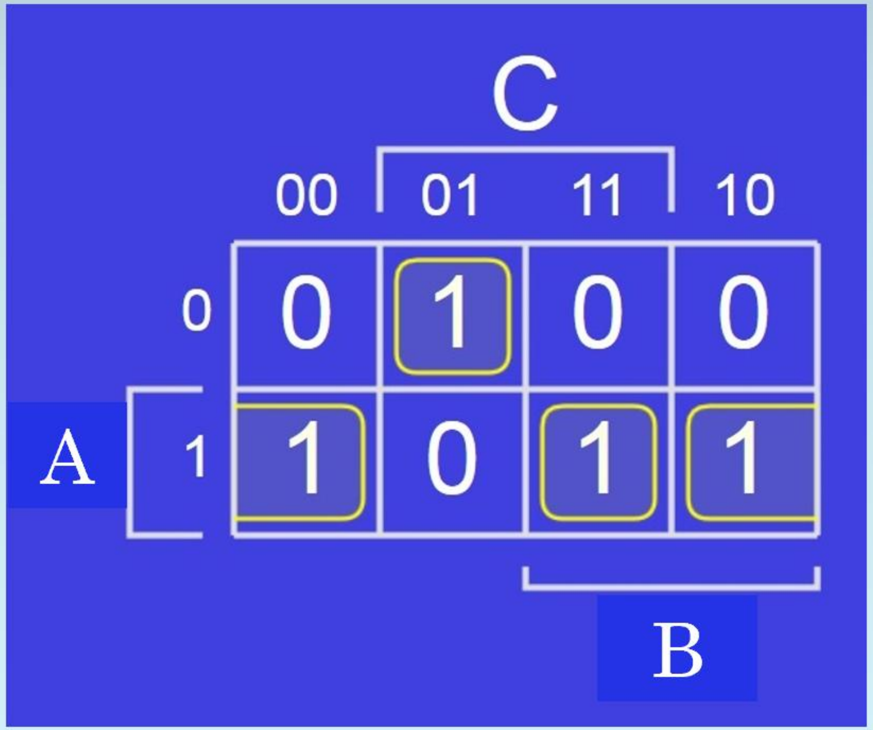




Let's consider a H(A,B,C) that we want to simplify

A	B	C	H(A,B,C)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1





Loops Can be 1, 2, or 4 Boxes Wide

So we **use Gray code order** on the boxes (one bit changes at a time).

Loops can be

- 1 box wide (a vertex)
- 2 boxes wide (an edge)
- 4 boxes wide (the face)

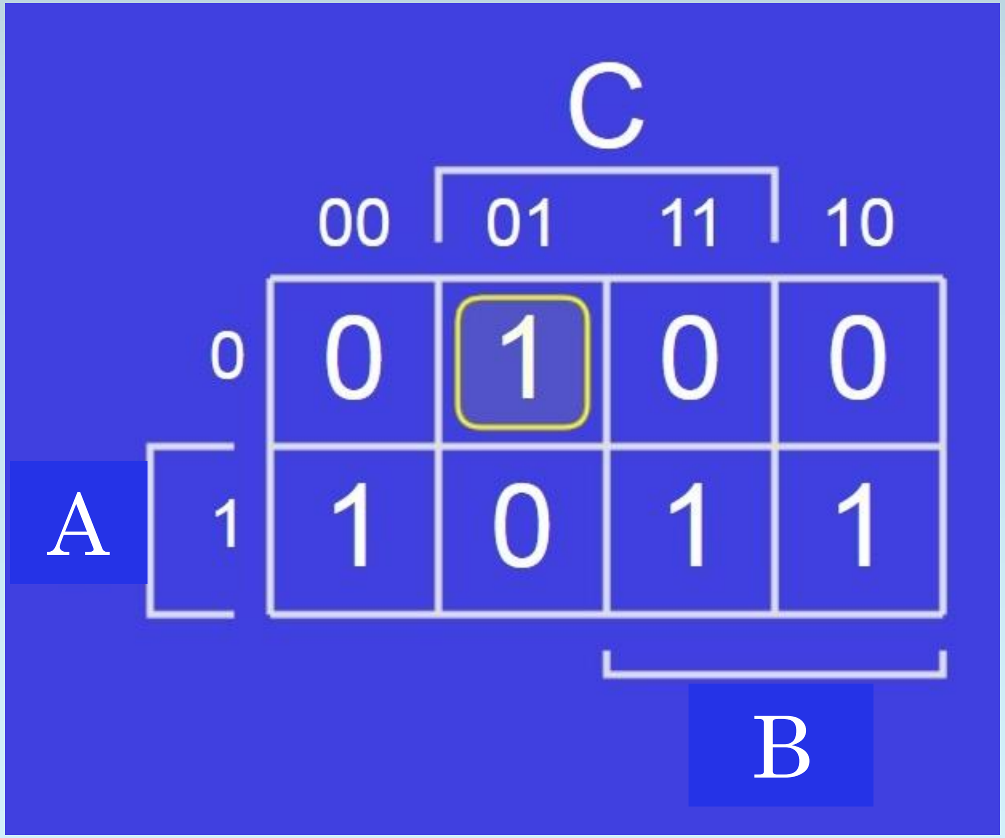
Loops cannot be 3 boxes wide, because 3 boxes do not correspond to an implicant (implicants are hypercube features).

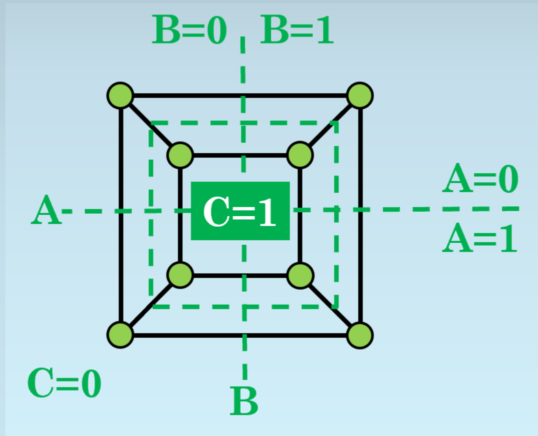
We Draw Function H(A,B,C) Using a 3-Variable K-Map

Here is a **3-variable K-map**.

Let's find a way to express **H(A,B,C)**.

Start by circling a 1.



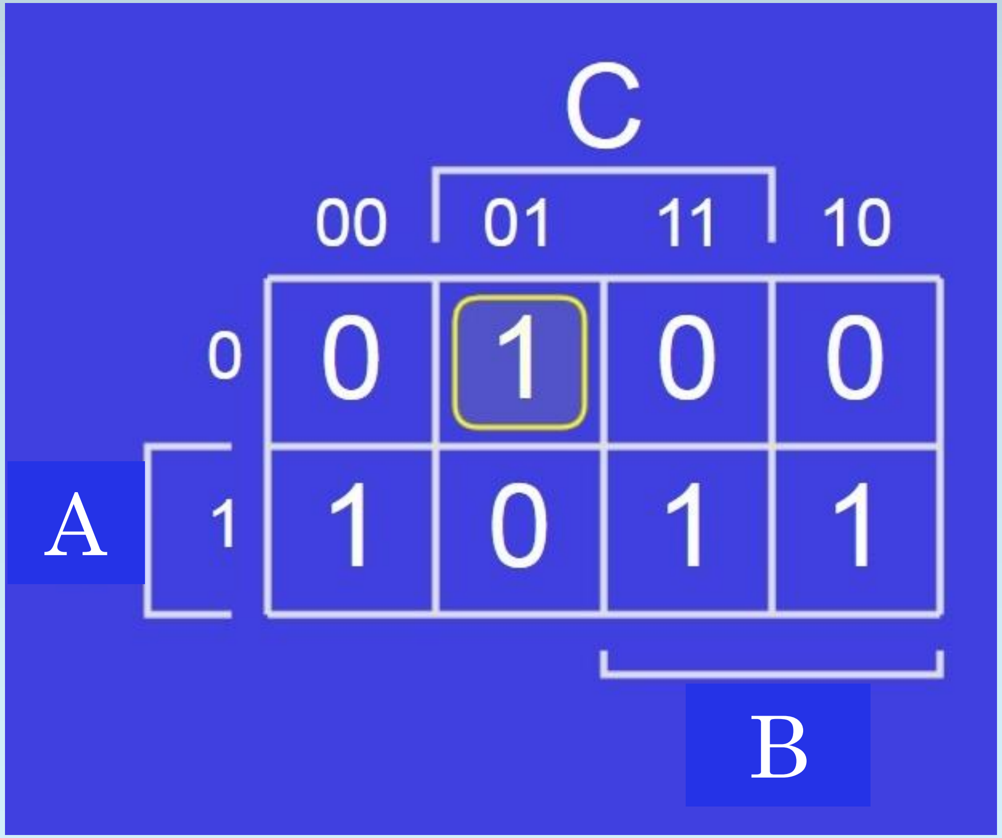


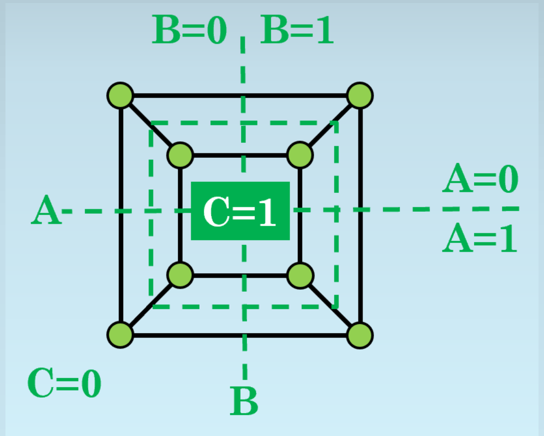
Some Minterms May Be Prime Implicants

The loop represents minterm **A′B′C**.

Is **A′B′C** a prime implicant of **H**?

Yes, since we cannot grow the loop left, right, nor downward.





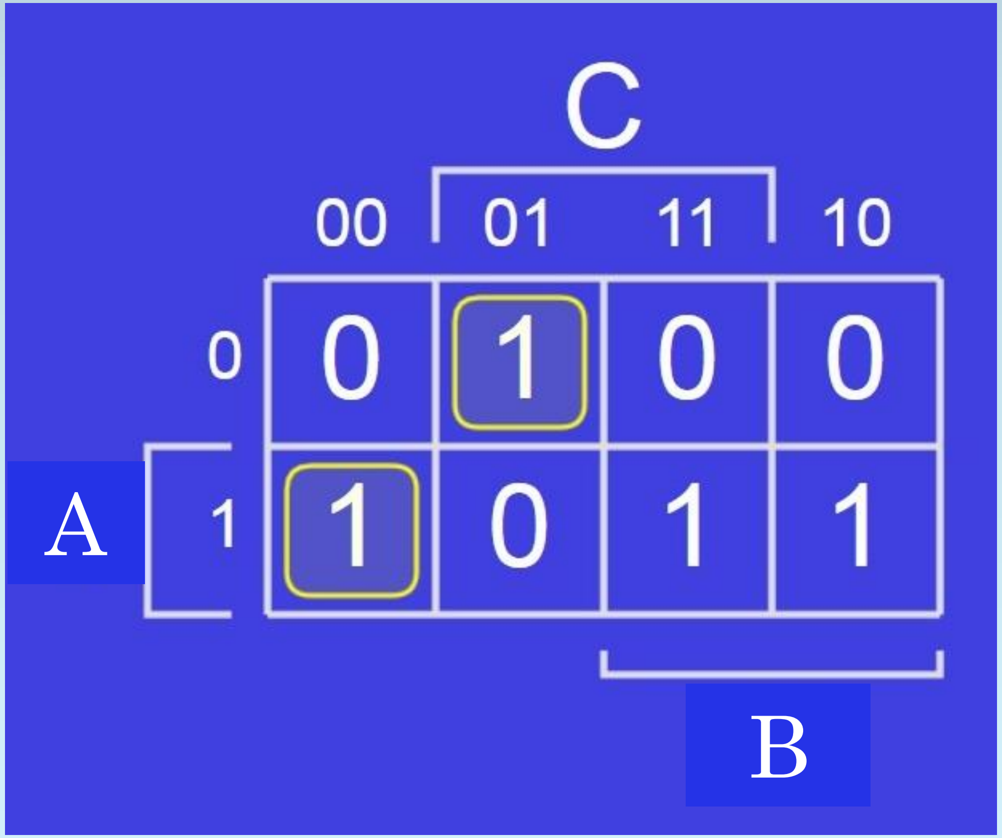
Don't Forget to Check for Wrapping

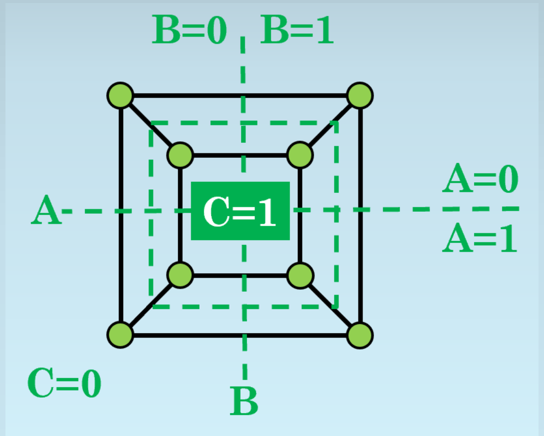
Choose another 1 to cover and circle it.

The new loop is the minterm **AB′C′** .

Is **AB′C′** prime for **H(A,B,C)**?

No, we can grow the loop to the left (wrap around).





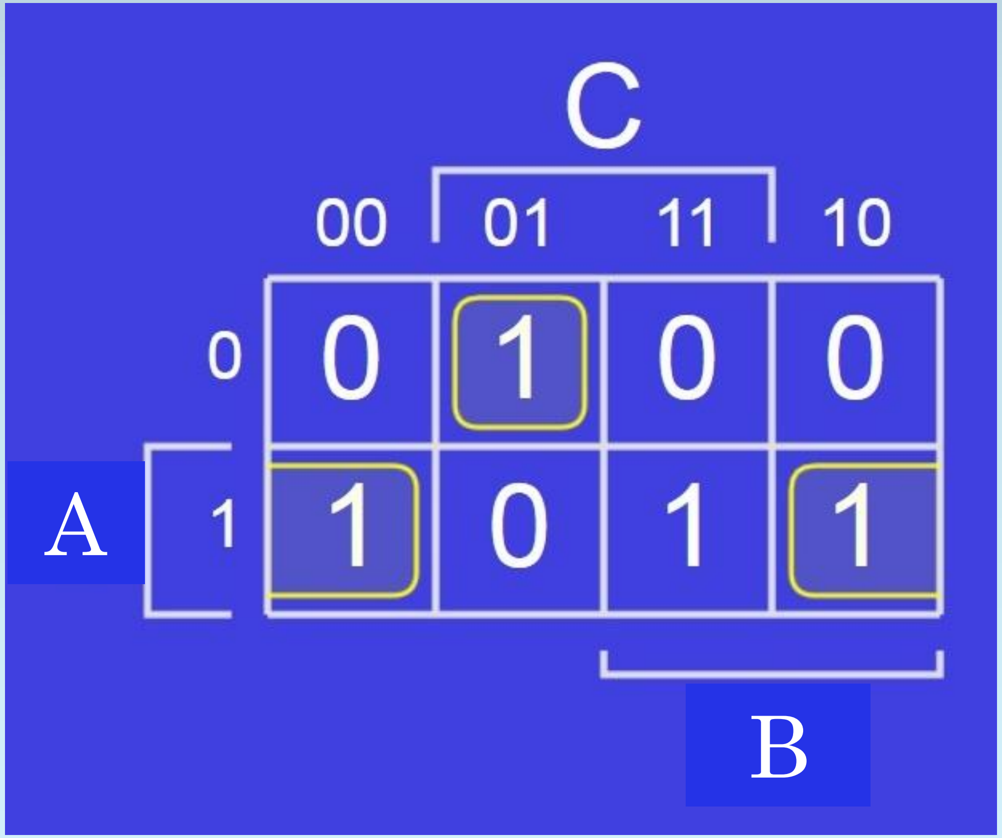
We Have Found a Second Prime Implicant

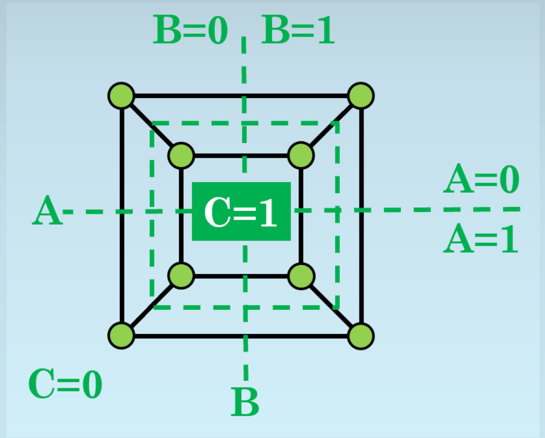
Grow the loop.

The new loop is **AC′** .

Is **AC′** prime for **H(A,B,C)**?

Yes. A loop cannot have three 1s, and we cannot include the 0 in the row.





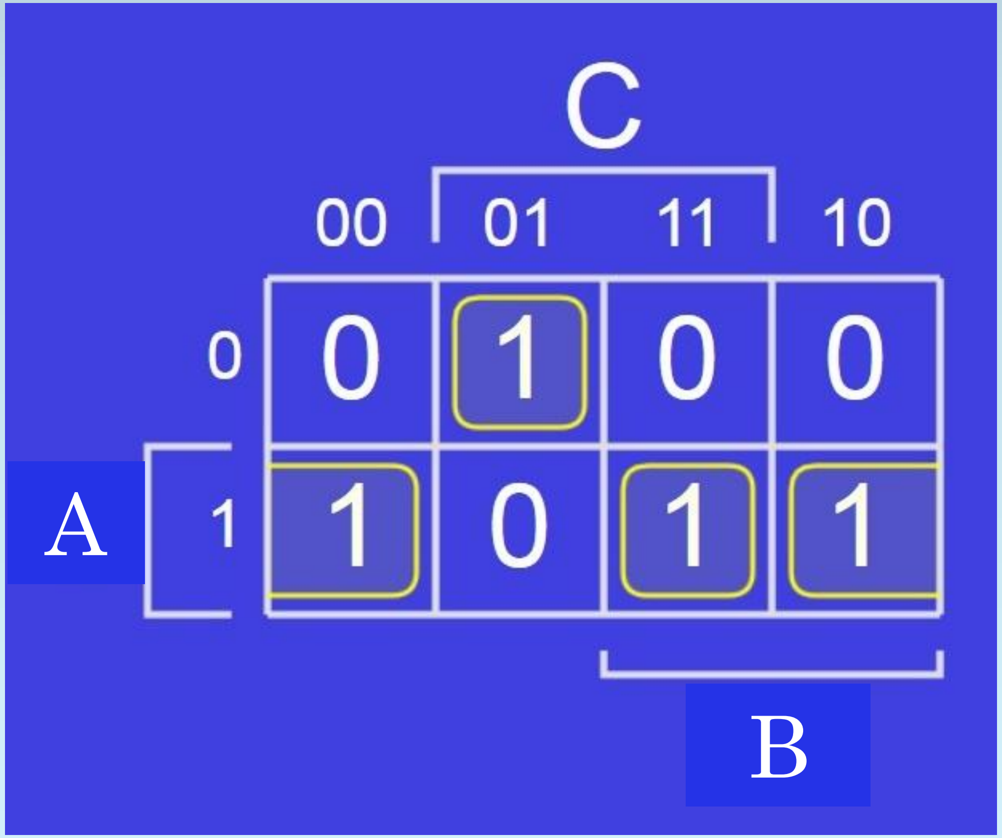
Keep Choosing Prime Implicants Until All 1s are Covered

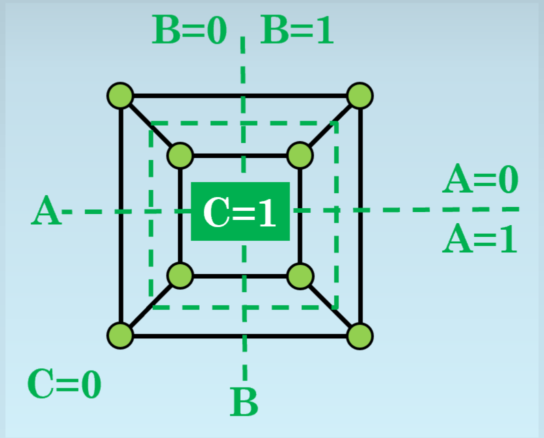
We still have another 1 to cover. Circle it.

The new loop represents minterm **ABC**.

Is **ABC** a prime implicant of **H**?

No, we can grow the loop to the right.





And We're Done: H(A,B,C) = A'B'C + BC' + AB

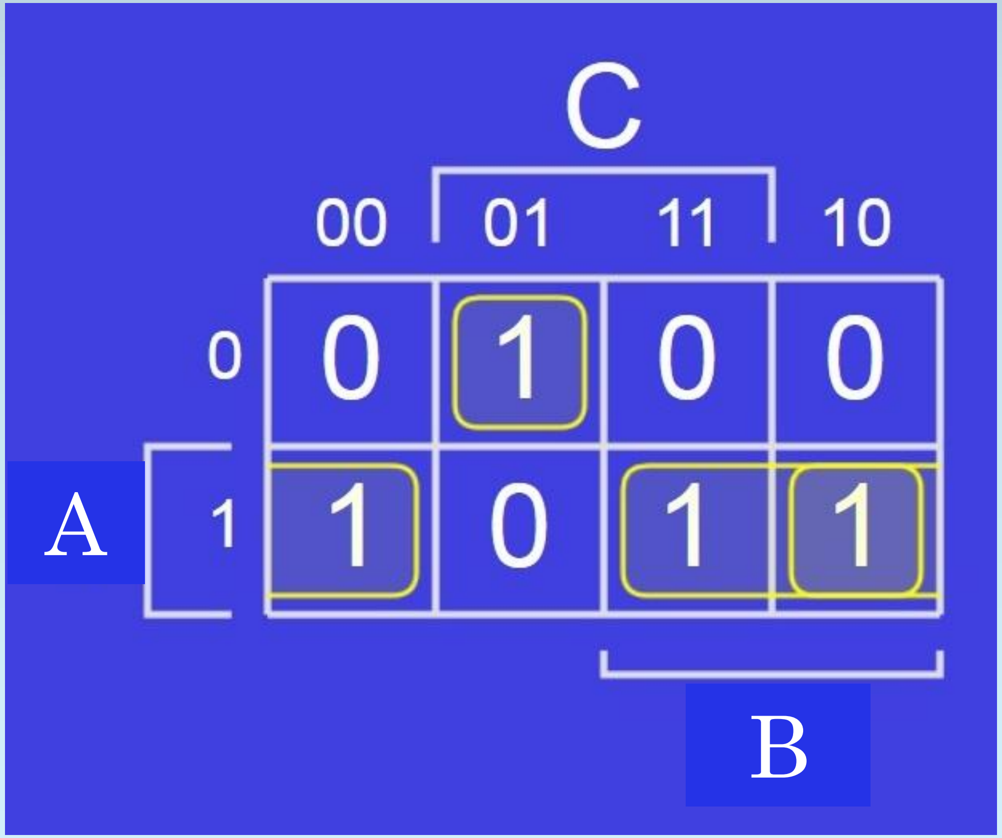
Grow the loop.

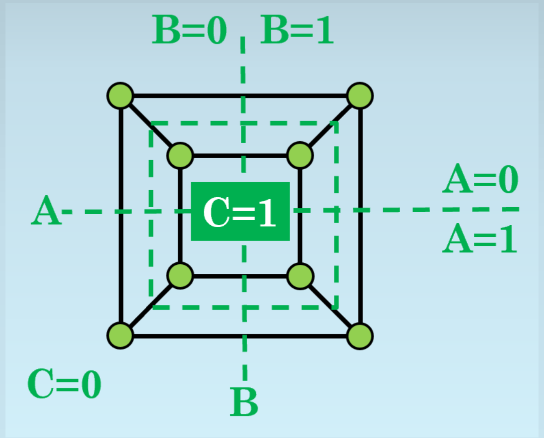
The new loop is **AB**.

Is **AB** prime for **H(A,B,C)**?

Yes.

So **H(A,B,C) = A'B'C+AC'+AB**





Last Class Example: F = AB'C + ABC' + ABC

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

BC

A	00	01	11	10
0				
1		1	1	1

F = AB + AC.

K-Maps Extend Nicely to Four Variables

Now you're excited?

Ok, on to 4 variables!

It's hard to draw the hypercube.

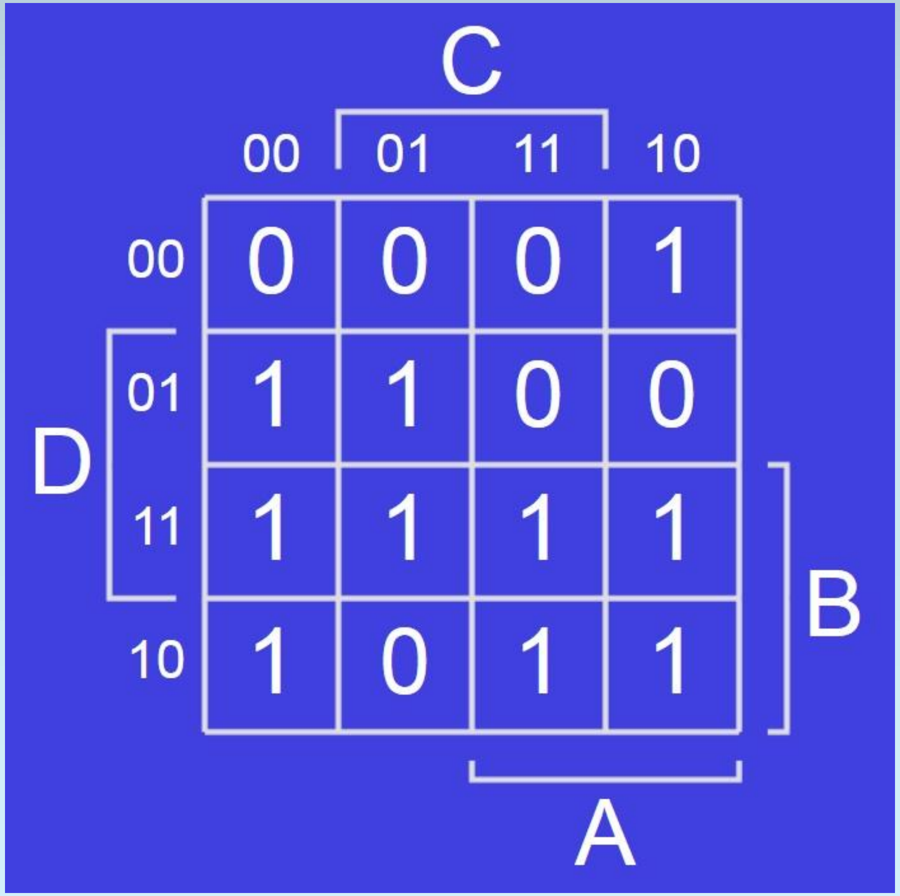
But the K-map is not so bad.

Remember:

- **Gray code order** in both directions.
- **1, 2, or 4-box loops** (no 3-box loops!).

Here's a 4-Variable K-Map

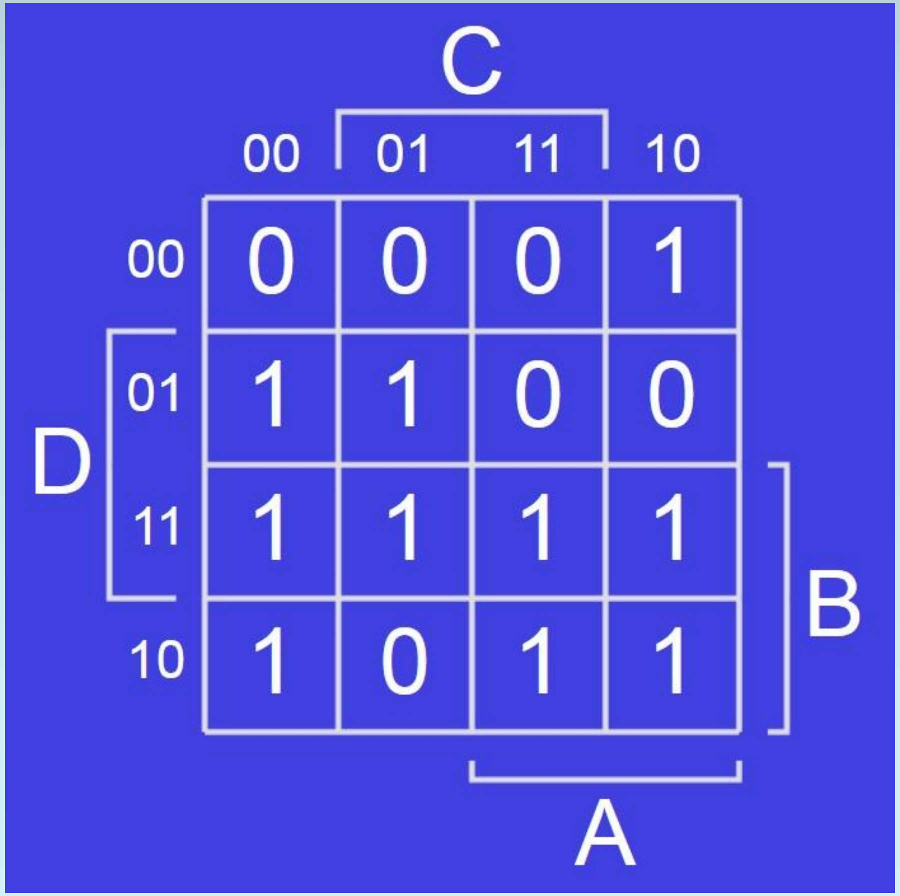
Here's how a **4-variable K-map** looks.



You can try it in the online tool?

Here’s a 4-Variable K-Map

Here’s how a **4-variable K-map** looks.



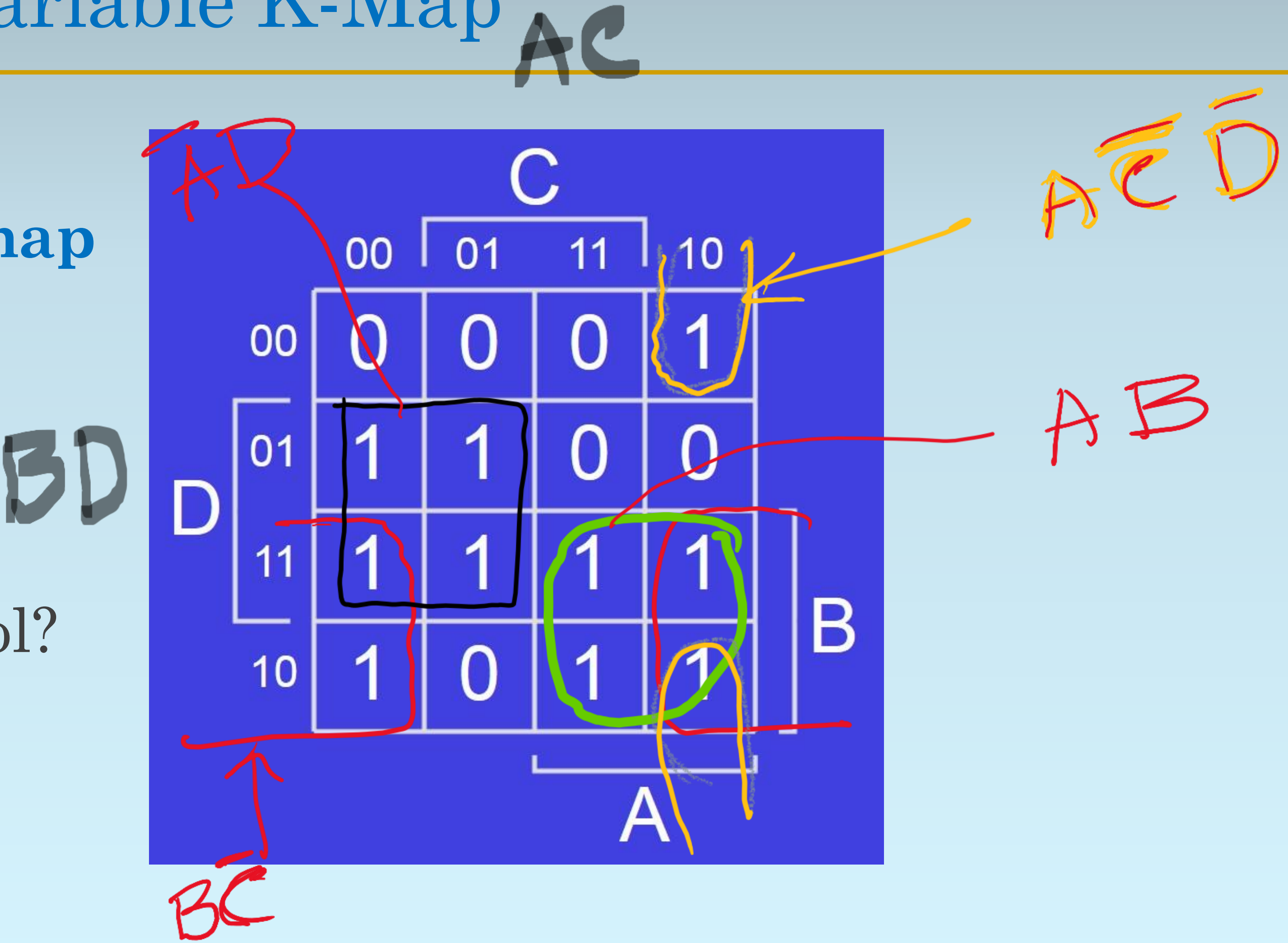
You can try it in the online tool?

Here's a 4-Variable K-Map

Here's how a **4-variable K-map** looks.

We won't solve this one now.

Want to try it in the online tool?



Goal: Minimal Number of Loops, Maximal Size per Loop

Your **goal** is to come up with

- a **minimal number of loops**
- of **maximal size** (all prime, of course).
- that together **cover all 1s** in the function.

If you do so, the **result will be optimal among SOP expressions* by our area heuristic** (for 4 or fewer variables).

*A POS expression might be better, as might an expression using XORs.

Another Example

In CMOS, we only have NAND and NOR

In SOP expression we have,

AND, followed by OR.

But in CMOS, we only have NAND and NOR.

What should we do??

DeMorgan's Laws Relate NAND/NOR to AND/OR

What do DeMorgan's Laws mean?

Here's one way to think about them:

- **(AB)' = A' + B'** NAND is the same as OR on the complements of the inputs.
- **(A+B)' = A'B'** NOR is the same as AND on the complements of the inputs.

Let’s Introduce Some Algebra

Demorgan Laws:

A	B	$\overline{A}$	$\overline{B}$	$A+B$	$\overline{A+B}$	$\overline{A}.\overline{B}$	$A.B$	$\overline{A.B}$	$\overline{A}+\overline{B}$
0	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	0	0	1	1
1	0	0	1	1	0	0	0	1	1
1	1	0	0	1	0	0	1	0	0

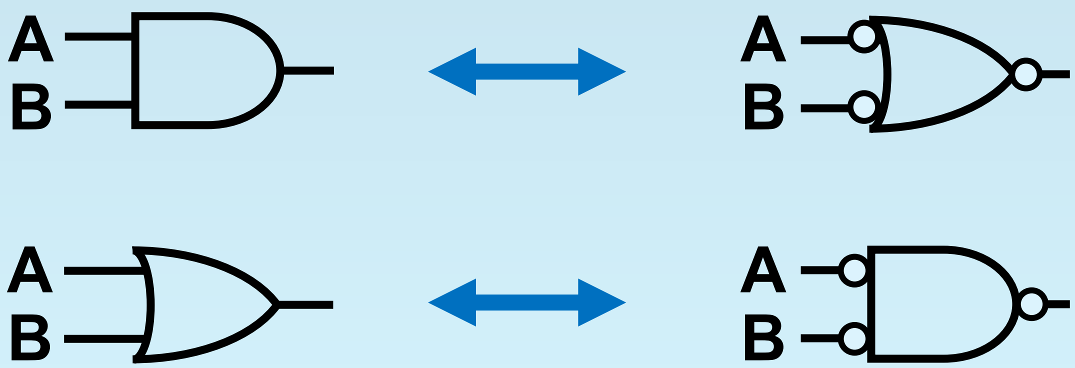
A Graphical Representation Can Be Useful, Too

Let's also think about them graphically.

Complement both sides first, so we have…

$$AB = (A' + B')' \qquad\qquad A+B = (A'B')'$$

and now we can draw gates…



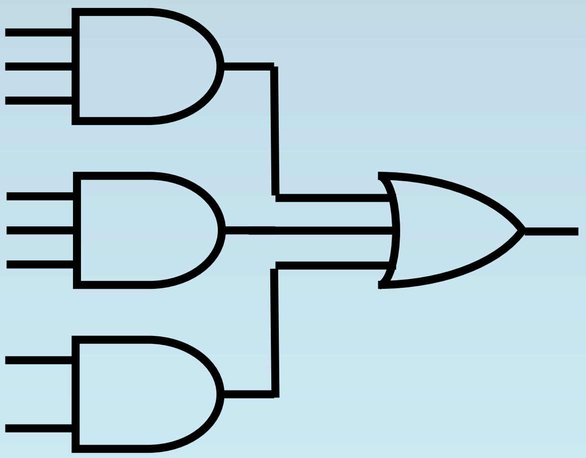
How Do We Draw an SOP Form? AND, then OR

What were we talking about?

Ah, speed of SOP forms.

SOP is AND followed by OR.

Something like this…

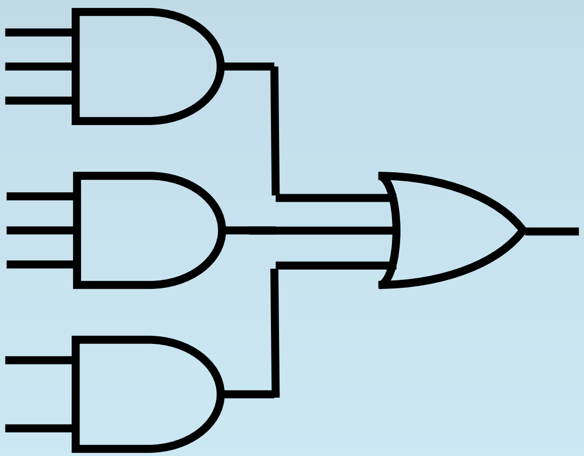


(with some number of AND gates, each with some number of inputs)

Apply DeMorgan's Laws Graphically

Use DeMorgan's law on the OR gate.

Replace it with a NAND with inverted inputs.

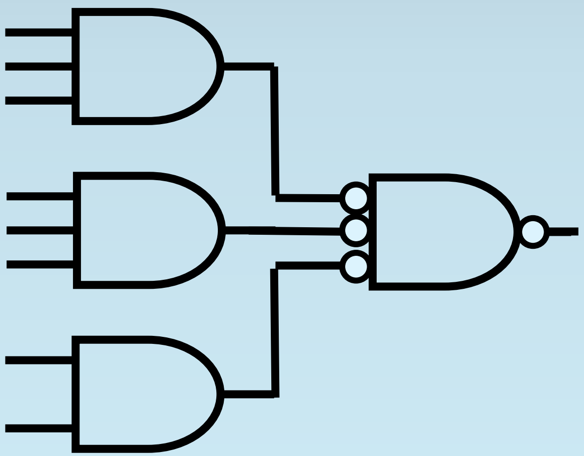


Apply DeMorgan's Laws Graphically

Use DeMorgan's law on the OR gate.

Replace it with a NAND with inverted inputs.

Remember that the **input bubbles mean inverters** (NOT).



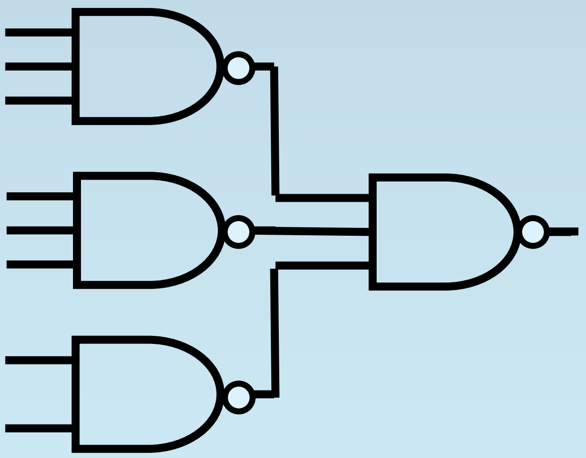
Now **slide them down the wires to the left** until they sit in front of the ANDs.

Apply DeMorgan's Laws Graphically

Use DeMorgan's law on the OR gate.

Replace it with a NAND with inverted inputs.

Remember that the **input bubbles mean inverters** (NOT).



Now **slide them down the wires to the left** until they sit in front of the ANDs.

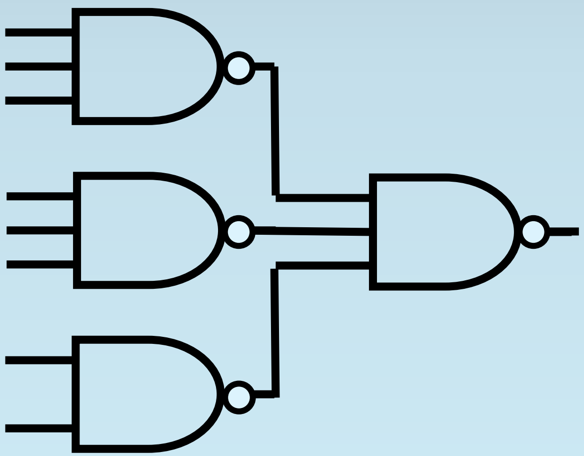
SOP Form Speed is Two Gate Delays

We didn't change the function of the circuit.

But now all of the gates are NAND gates.

So **we can build any SOP function using two levels of NAND**.

And the speed? **Two gate delays.**



SOP and POS Forms Give Us Two-Level Logic

We can use two levels of NANDs to build any SOP expression.

We refer to this approach as **two-level logic**.

For a POS expression

- **one can do exactly the same thing**
- replacing OR followed by AND
- **with NOR followed by NOR**.

So **any POS expression also requires two gate delays** (again, assuming that complemented inputs are free).

Use a K-Map to Find POS Expressions

But **how can we find a POS form?**

Again, **use a K-map**.

1. Given a function **F**, draw a K-map for **F'**.
2. Use K-map to **find an SOP form for F'**.
3. **Complement the result** to find **F**
   - and apply DeMorgan's laws a few times,
   - **complement** of SOP form **is POS form**.

In Practice, Form Loops Around 0s to Find POS

In practice, **just circle 0s instead of 1s**.

Recall that a box in a K-map
- when filled with a 1
- corresponds to a **minterm**.

The same box
- when filled with a 0
- corresponds to a **maxterm**
- an expression that produces exactly one 0 row in its truth table.

Complement Literals When Reading POS Factors

But be careful: the **maxterm** has all variables complemented relative to the **minterm**.

For example,

- a box corresponding to **minterm ABC'** (equal to 1 when **A=1** and **B=1** and **C=0**)
- corresponds to **maxterm A' + B' + C** (equal to 0 when **A=1** and **B=1** and **C=0**)

SOP and POS Forms Give Us Two-Level Logic

To **find a POS form** that has optimal area (among POS forms),

- **follow the same approach** as before,
- but instead of drawing loops around 1s,
- **draw loops around 0s**.

Again, **do not forget to complement the literals relative to their form for implicants!**

(And write each loop as a sum, not as a product.)

Which Form is Better? Solve Both and Compare

Which gives better area, SOP or POS?

That depends on the function.

Solve both ways and compare.

You will have some experience finding POS forms in discussion section.

You can also use the online tool, but the exercises are not as direct as for SOP.