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

ECE 120: Introduction to Computing

Boolean Expression Terminology

Let's Review and Define Some New Terms

literal a variable or its complement

examples: A, A', B, B', C, C'

sum several terms ORed together

examples: $A + B$, $AB + B(C + D) + A'C$,

$A'B' + D(A \oplus B)(C + A')$

product several terms ANDed together

examples: AB, $(A + B)(B + CD)(A' + C)$,

$(A' + B')(D + (A \oplus B) + CA')$

Minterms Were Useful for Proving Logical Completeness

minterm on N inputs

a product in which each variable or its complement appears exactly once (no other factors)

examples: **AB'**, **A'B**, **AB** (on inputs **A**, **B**)

AB'C, **AB'C'**, **A'BC'**

(on inputs **A**, **B**, **C**)

A Maxterm Produces a Function with One Zero Row

maxterm on N inputs

a sum in which each variable or its complement appears exactly once (no other terms)

examples: **(A + B')**, **(A' + B)**, **(A + B)**

(on inputs **A**, **B**)

(A + B' + C), **(A + B' + C')**,

(A + 'B + C')

(on inputs **A**, **B**, **C**)

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 sum 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?

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** $\rightarrow$ X can fly.)

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

(X scored **above 125%** $\rightarrow$ 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**.

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.

B'C is not an implicant.

AC is an implicant.

AB' 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

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

$$F = AC + AB + AC$$

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

$$F = AC + AB$$

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.

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.

List All Implicants for One Variable A

Let's try a different approach.

Start with functions of one variable, **A**.

How many implicants are possible?

Remember:

- There are only four functions on **A**!
- We only consider products of literals.

`A` **`A'`** **`1`**

(**`1`** is the product of zero literals.)

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**.

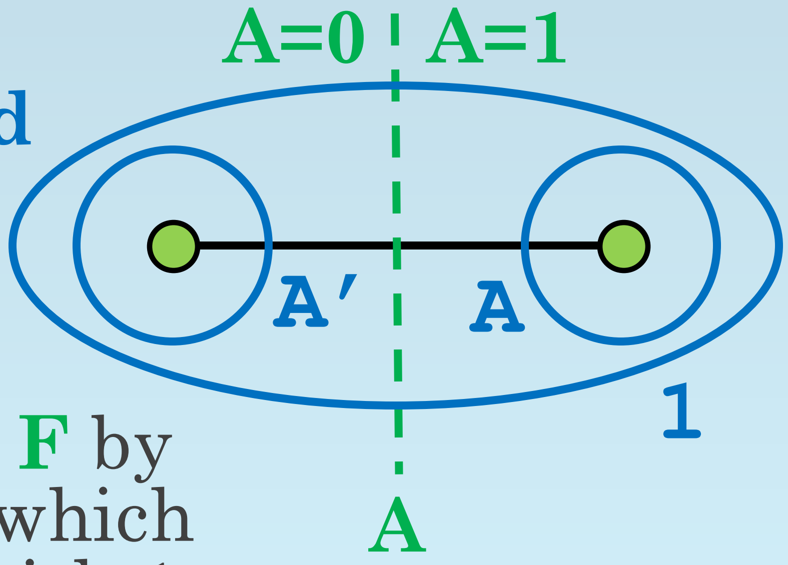
Each vertex in the hypercube corresponds to one combination of the **N** inputs.

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

Implicants for N=1 Correspond to Vertices and Edge

With **N = 1** (one variable, **A**), a hypercube is just a line segment with two vertices.

The three possible **implicants correspond to** the **two vertices** and the **one edge** of the hypercube.



If we write the values of **F** by the vertices, we can see which implicants are covered with 1s.

We Draw Function F(A) Using a 1-Variable K-Map

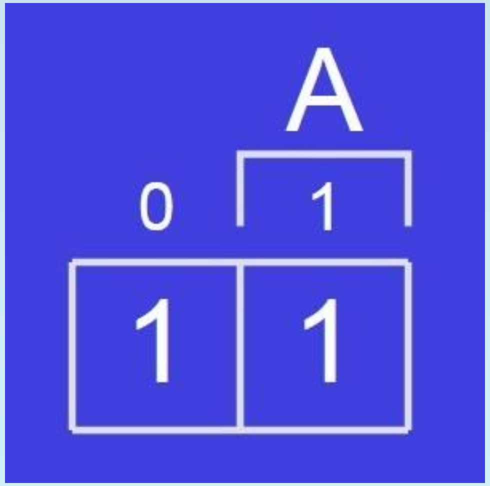
Instead of drawing a line segment, we can draw two boxes, as shown below.

We call this approach a **Karnaugh map** (**K-map**) on 1 variable.

The left box corresponds to **A = 0**, and the right corresponds to **A = 1**.

Each box represents

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



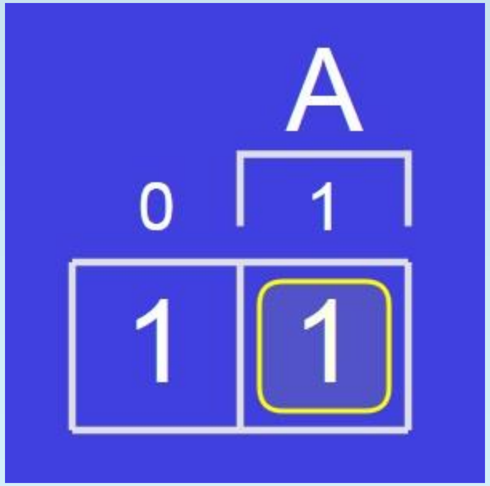
We Draw Function F Using a 1-Variable K-Map

We can mark implicants of **F** by **circling boxes that contain 1s**.

Here, we show a **loop** around the box corresponding to the **implicant A**.

To check whether an implicant is prime, we consider **growing the loop** to contain more boxes.

A circle that cannot grow is a prime implicant of **F**.



We Draw Function F Using a 1-Variable K-Map

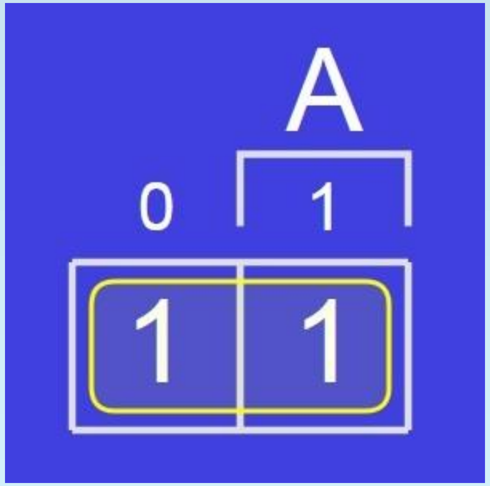
For the function **F** shown, we can grow the loop to contain both boxes.

The loop is now as big as possible (the full K-map!), so it cannot grow further.

The result (the **implicant 1**) **is a prime implicant of F**.

So **F(A) = 1**.

Feel excited?



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…

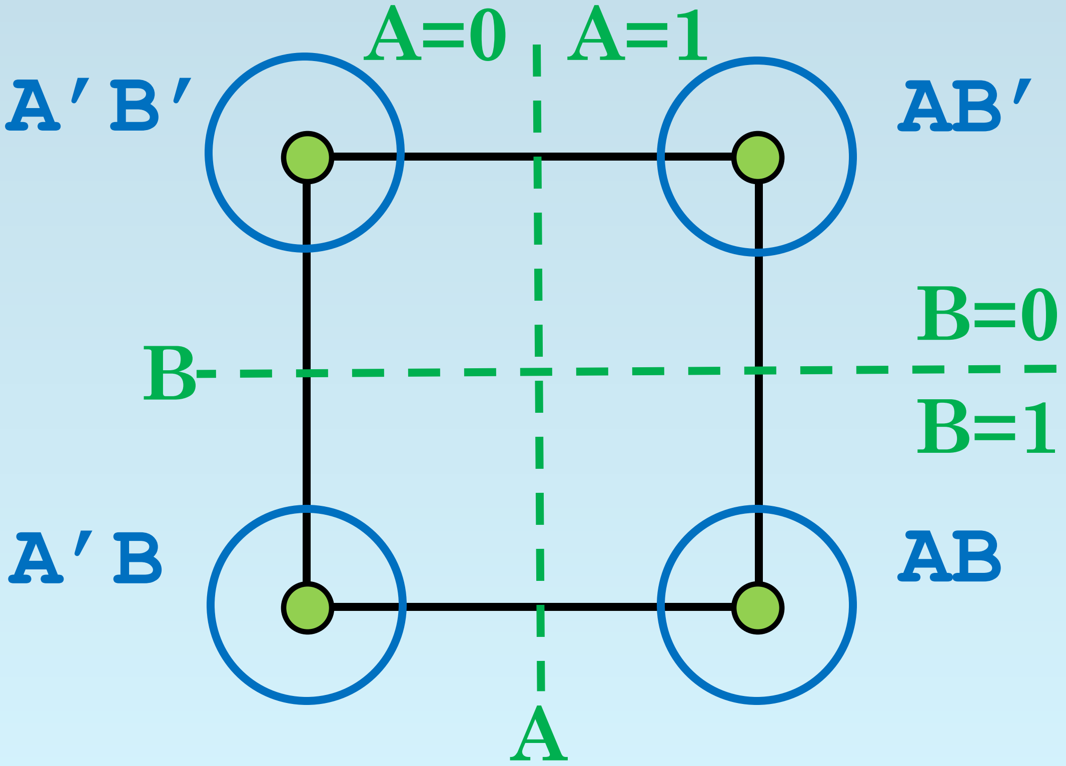
A A′ B B′

And, of course …

1

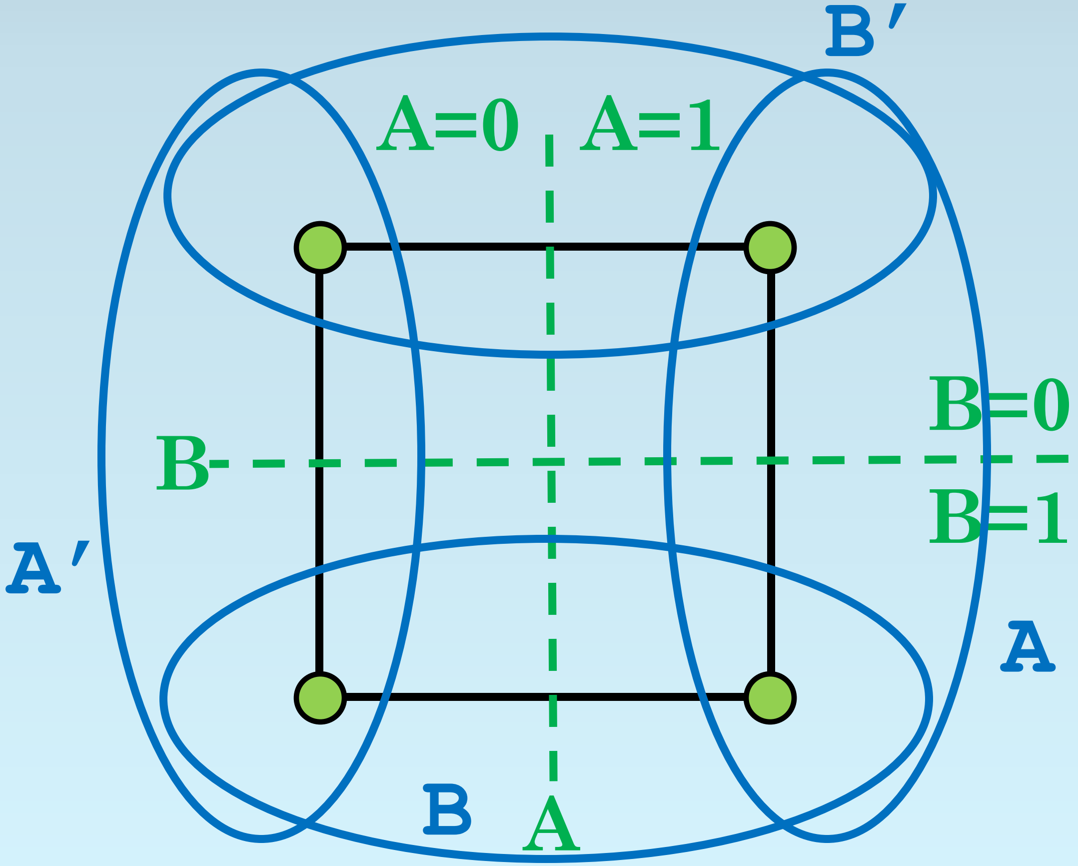
Minterms Correspond to Vertices

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



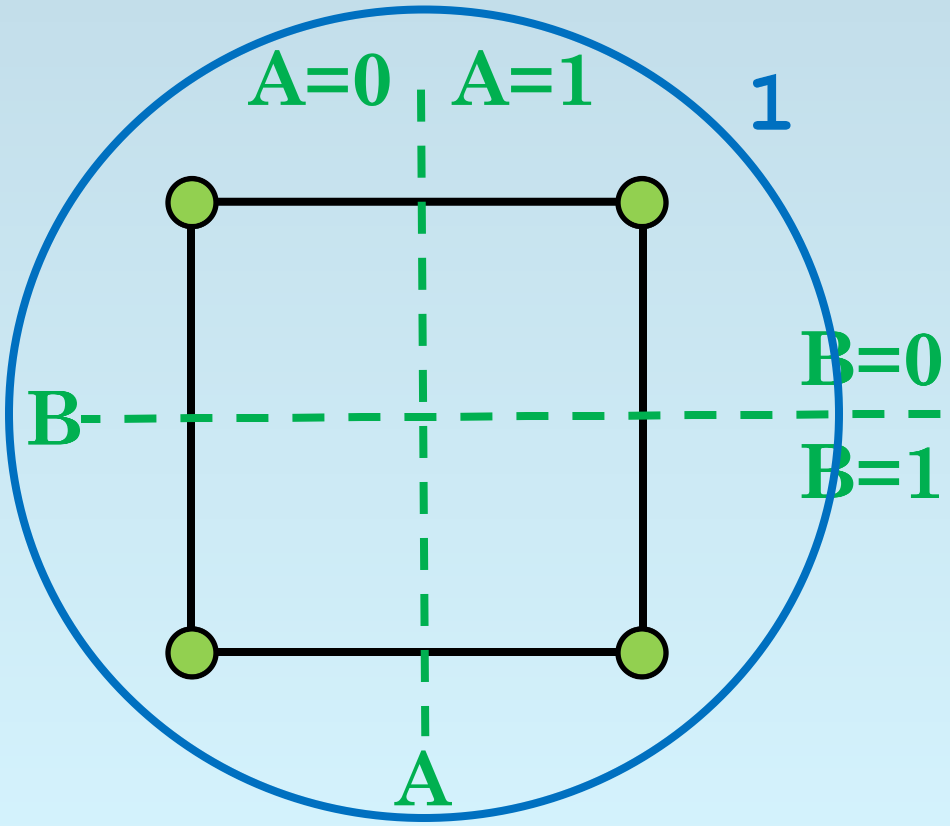
Single-Literal Implicants Correspond to Edges

Edges include both values of one variable.



The Implicant 1 Corresponds to the Face/Square

The face includes both values of both variables.

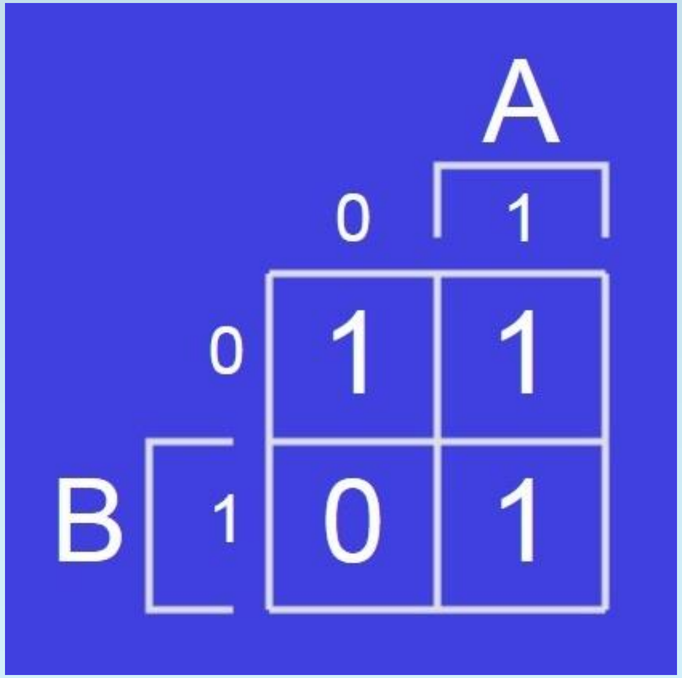


We Draw Function G(A,B) Using a 2-Variable K-Map

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

Again, each box represents

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



Process for Finding G(A,B) Using a K-Map

Now the problem is more interesting.

We want to **find the largest loops**

- with power-of-2 edge lengths (1 or 2)
- **that together cover all 1s** in **G**.

Why?

- A **loop that can't grow is a prime implicant** of **G**.
- If we cover all 1s, **the sum of the implicants gives the function G**.

To Find G, Start by Picking a 1 and Circling It

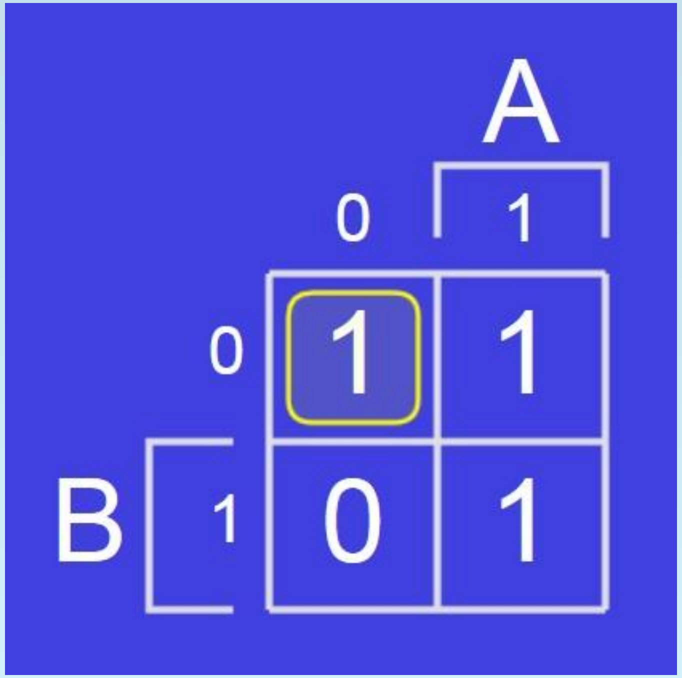
Start by picking a 1 and circling it.

The minterm **`A'B'` is an implicant of G**.

But it's **not a prime implicant of G**.

We cannot grow the loop downward (cannot cover a 0—that would not be an implicant).

We can **grow the loop to the right**…



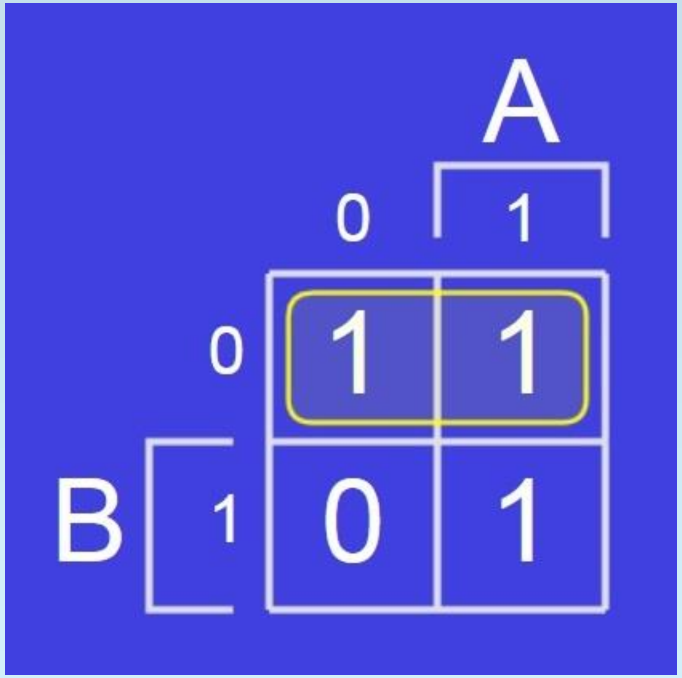
Grow the Loop Until We Get a Prime Implicant

Let's grow the loop.

The loop now represents **`B'`, which is a prime implicant of G**.

But we didn't cover one of the 1s in **G** yet.

We **need a second loop**.



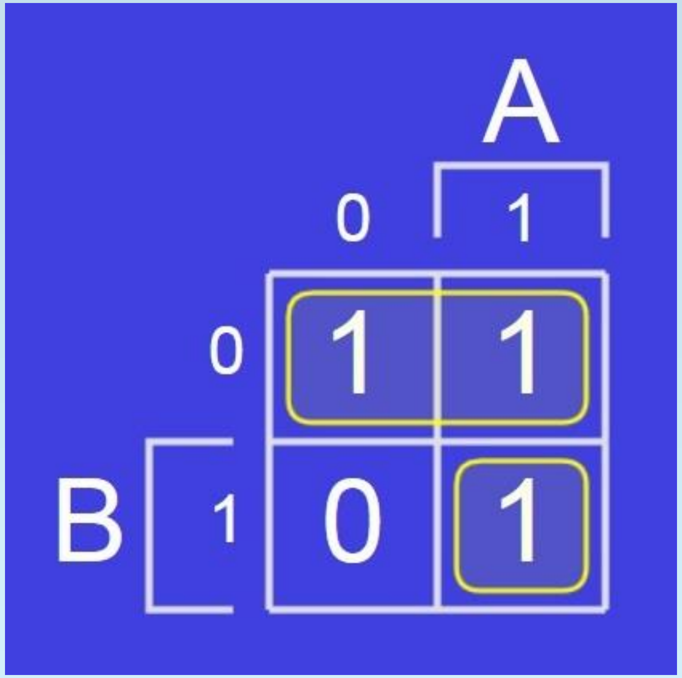
Start a Second Loop by Circling an Uncovered 1

The new loop represents the minterm **AB, which is an implicant of G**.

But it's **not a prime implicant of G**.

We cannot grow the new loop to the left.

We can **grow the new loop upward**…



Again, Grow the Loop Until We Get a Prime Implicant

Let's grow the loop.

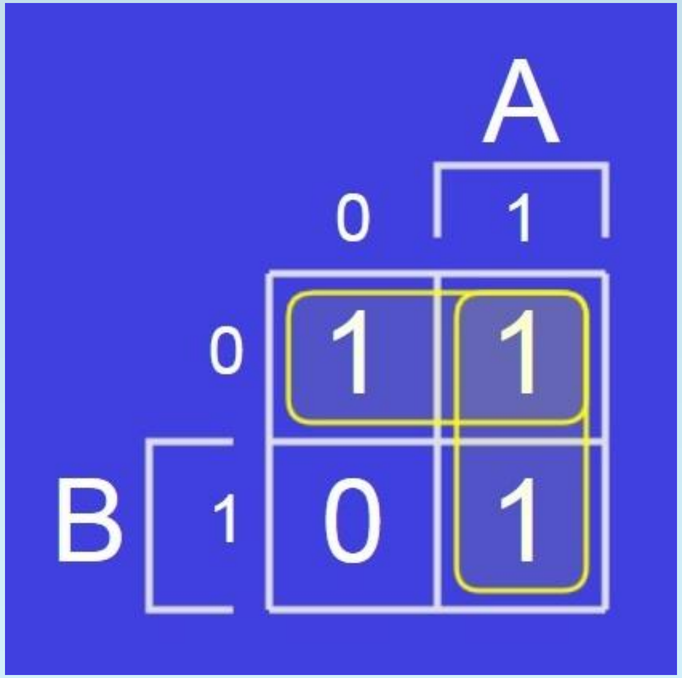
The loop now represents **A, which is a prime implicant of G**.

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

So we can write

$$G(A,B) = B' + A$$

Now 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.

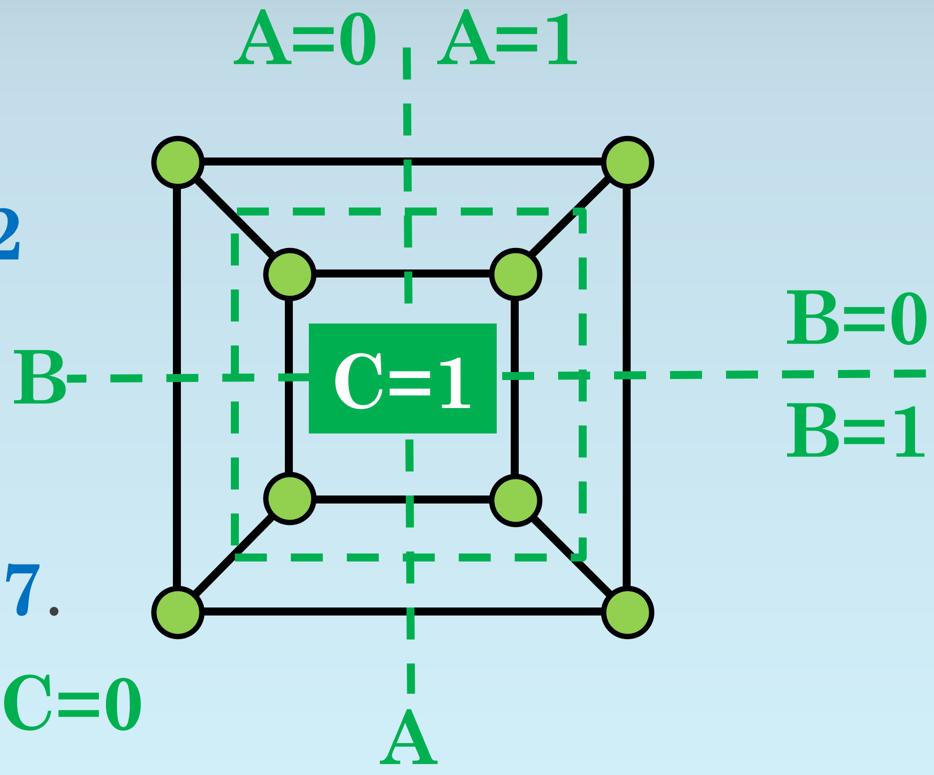
# of vertices? 8

# of edges? 12

# of faces? 6

# of cubes? 1

So the **total is 27**.



Notice a Pattern? 3^N Implicants on N Variables

N = 1 gives **3** implicants.

N = 2 gives **9** implicants.

N = 3 gives **27** implicants.

Maybe **N** gives $\mathbf{3^N}$ implicants?

Why?

For each input variable, we have **three choices**:

- include the variable
- include the complemented variable, or
- leave the variable out.

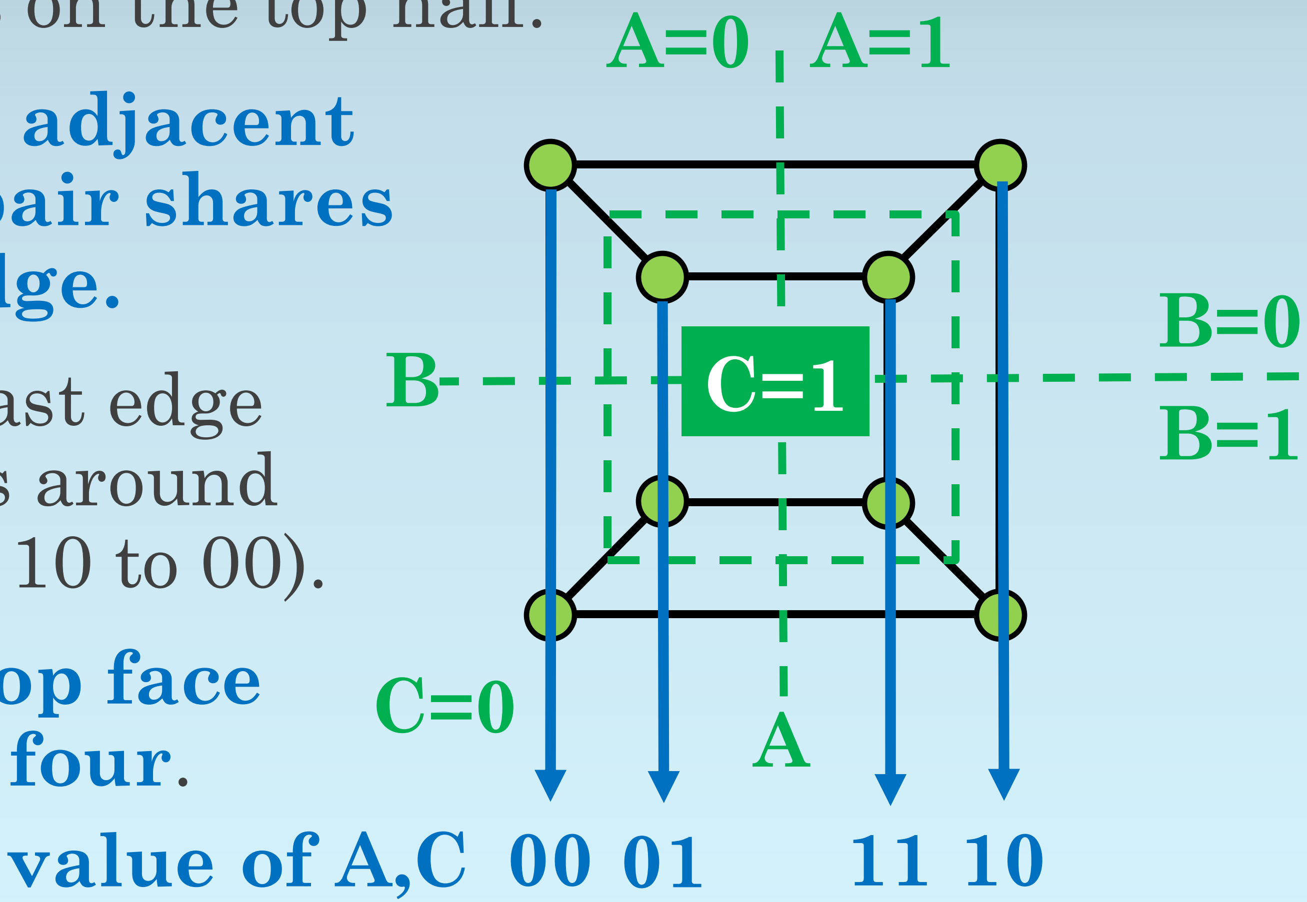
How Can We Draw Boxes for the Cube?

Focus on the top half.

Each adjacent A,C pair shares an edge.

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

The **top face is all four.**



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)

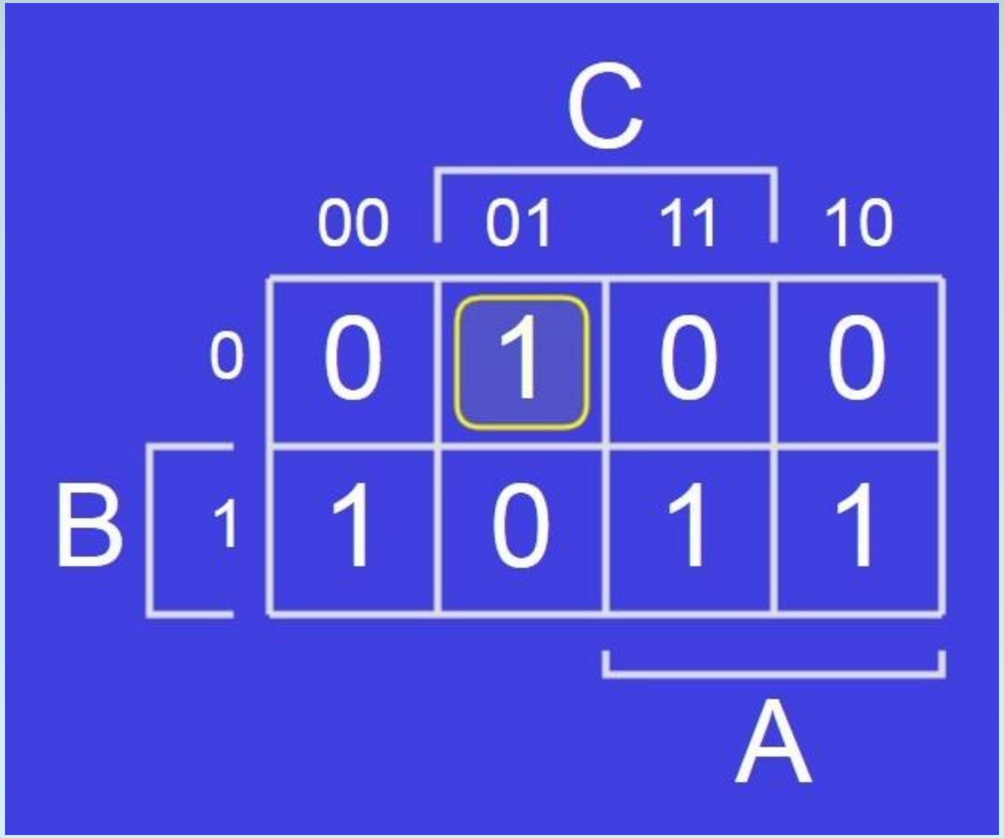
Loops cannot be 3 boxes wide, because 3 boxes do not correspond to an implicant (implicants are hypercube features).

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.

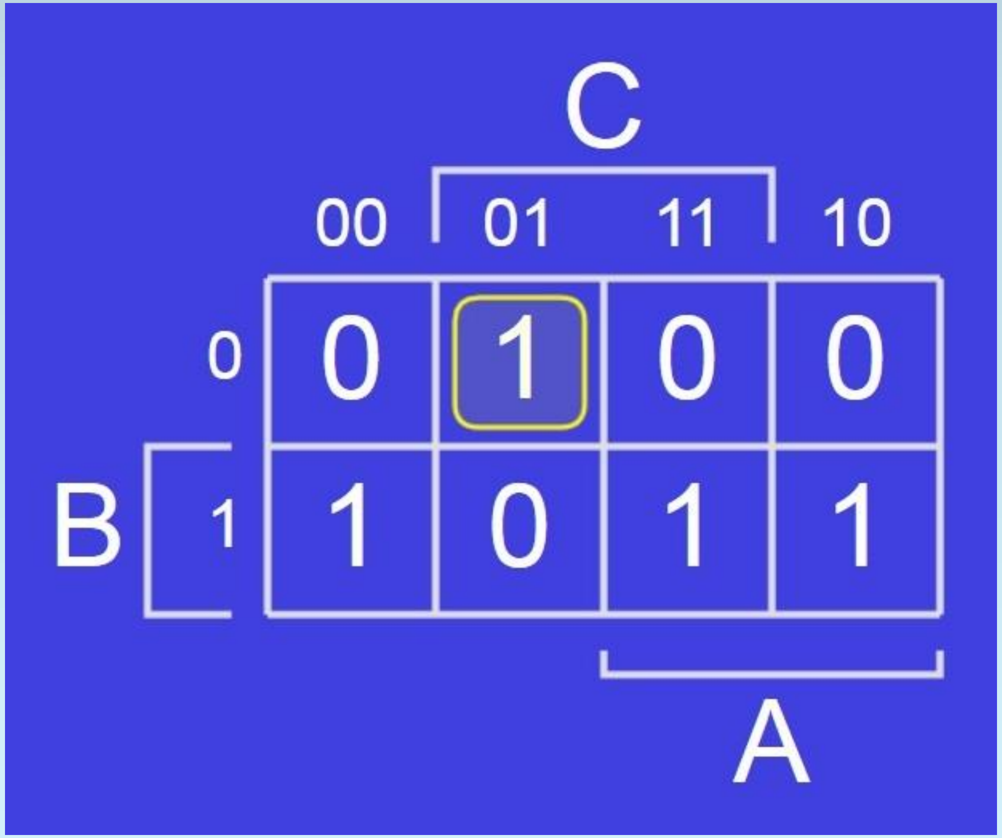


Some Minterms May Be Prime Implicants

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.



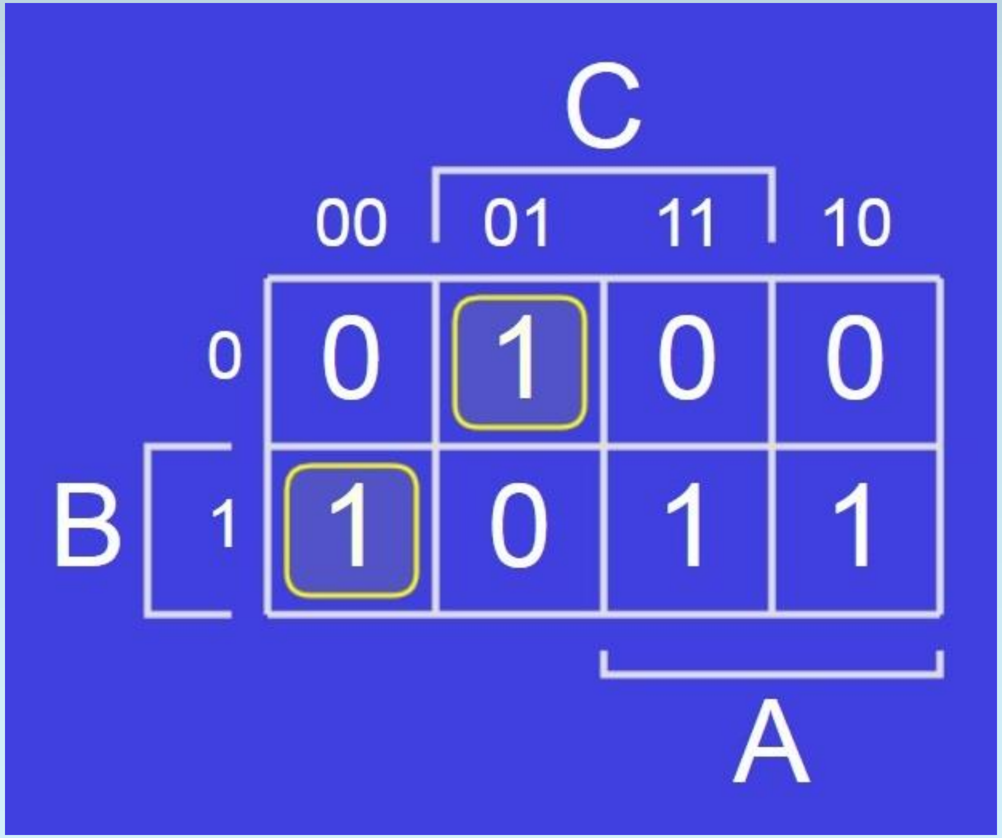
Don’t Forget to Check for Wrapping

Choose another 1 to cover and circle it.

The new loop is the minterm **A′BC′** .

Is **A′BC′** prime for **H(A,B,C)**?

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



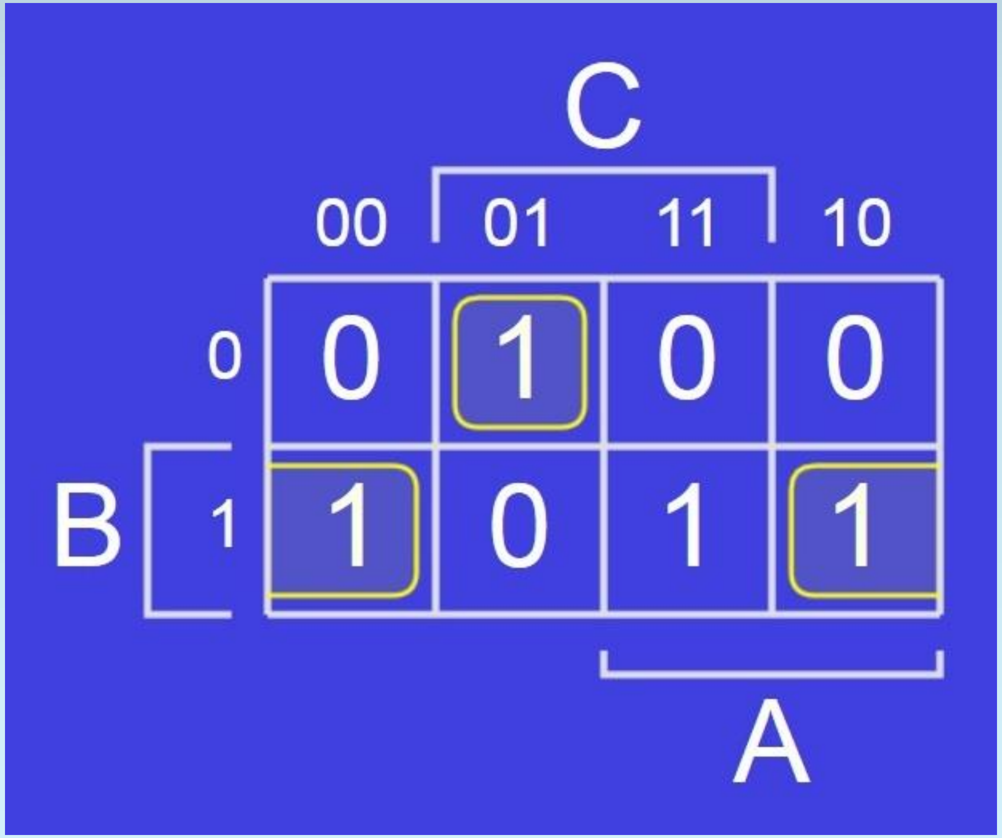
We Have Found a Second Prime Implicant

Grow the loop.

The new loop is **BC′**.

Is **BC′** prime for **H(A,B,C)**?

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



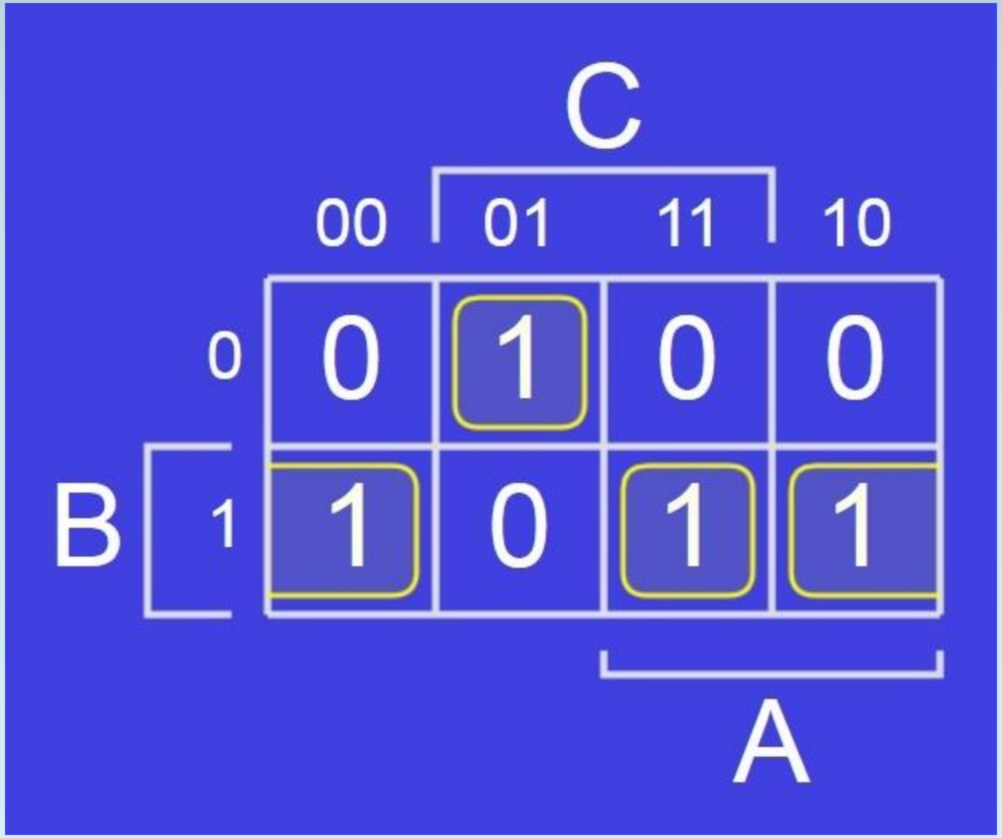
Keep Choosing Prime Implicants Until All 1s are Covered

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

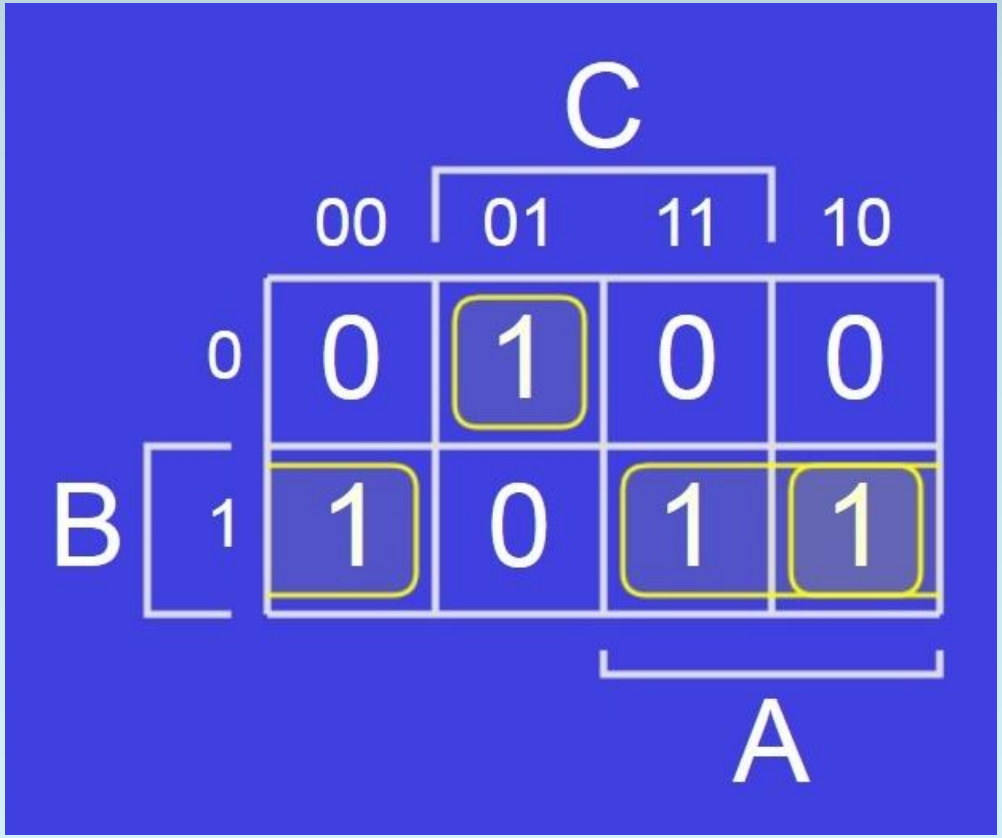
Grow the loop.

The new loop is **AB**.

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

Yes.

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



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!).

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.

Considerations for Optimizing with K-Maps

Sometimes you end up with loops that aren't needed. If all of a loop's 1s are covered by other loops, you can remove the loop.

To make the process faster,

- try to **start by covering 1s for which you need make no choices**
- (1s for which all directions with adjacent 1s can be included in one big loop).

But you may have to make choices, and **there can be more than one optimal SOP form**.

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?