Logic Circuits

...

Written by Rachel J. Morris, last updated 3/1/17

Chapter 3.5

This Chapter:

1) Using Boolean algebra with logic gates

2) Techniques to simplify circuits

3) Karnaugh maps

Data representation

Chapter 3.5 is about how Boolean algebra relates to computer circuits.

This is based on how computers store data as binary - 0's and 1's. This is true for representation of letters and numbers, as well as representing different kinds of commands that a computer can run.

Data representation

Perhaps you've seen an ASCII table at some time:

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

From https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg

Data representation

Perhaps you've seen an ASCII table at some time:

ASCII TABLE

'A' = 65

'a' = 97

Backspace = 9

Decimal	Hex	Char	Decimal	Hex	Char			Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99		
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100		
5	5	[ENQUIRY]	37	25	%	69	45	E	101		
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102		
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11			43	2B	+	75	4B	K	107	6B	k
12			44	2C	,	76	4C	L	108	6C	l
13			45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81					
18	12	[DEVICE CONTROL 2]	50	32	2	82					
19	13	[DEVICE CONTROL 3]	51	33	3	83					
20	14	[DEVICE CONTROL 4]	52	34	4	84					
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85					
22	16	[SYNCHRONOUS IDLE]	54	36	6	86					
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87					
24	18	[CANCEL]	56	38	8	88					
25	19	[END OF MEDIUM]	57	39	9	89					
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Notice that numbers are given as codes for letters, numbers, and even computer commands that don't have a graphical representation have codes.

These number codes are further translated into binary somewhere along the line.
'A' is 65, and in binary it is 0100 0001.

From https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg

Data representation

Once we get into the Assembly level, the lowest level programming we can do (except for programming directly in binary), our binary representations look like this:

Addition
Bitfields:

MIPS Assembly code: `ADD rd, rs, rt`

000000	rs	rt	rd	00000	100000

Subtraction
Bitfields:

MIPS Assembly code: `SUB rd, rs, rt`

000000	rs	rt	rd	00000	100010

Branch if equal
Bitfields:

MIPS Assembly code: `BEQ rs,rt,offset`

000100	rs	rt	offset

Data representation

So when you write Assembly code, it has a 1:1 translation to binary. And with higher-level languages, one command may translate to several Assembly commands.

Example MIPS assembly code

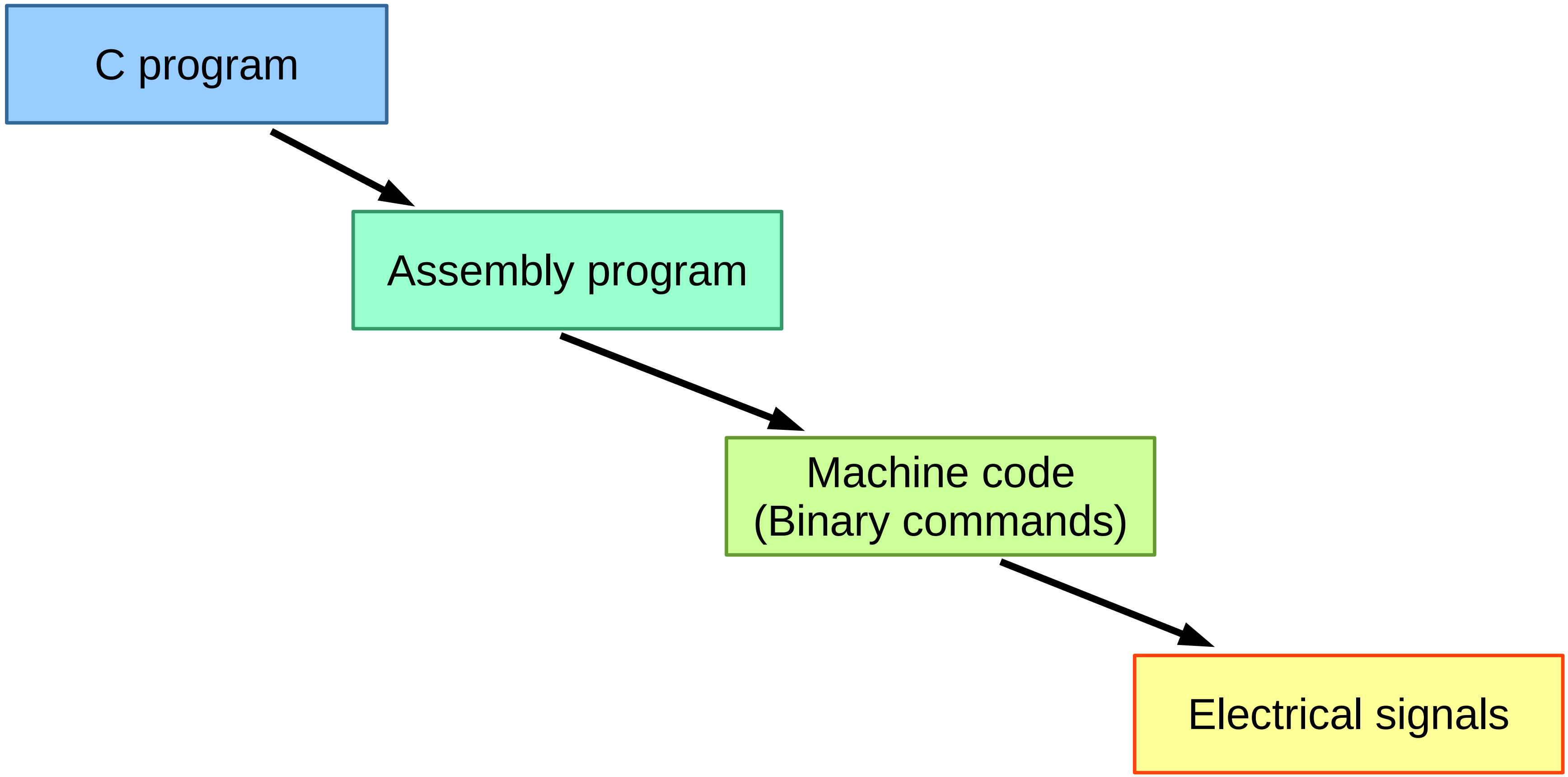
```
main:
        # load values into "temp" registers
        li $t0, 2               # $t0 = 2
        li $t1, 5               # $t1 = 5

        # add unsigned integers
        addu $s0, $t0, $t1      # $s0 = $t0 + $t1

        # display value of $s0
        li $v0, 1               # prepare to output number
        move $a0, $s0           # load parameter
        syscall                 # display
```

Data representation

So everything ends up boiling down to *binary* electric signals…

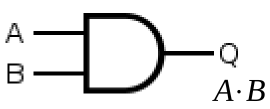


Logic Gates

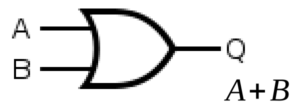
Logic gates

When designing circuits, some of the tools we have are logic gates to handle and, or, and not.

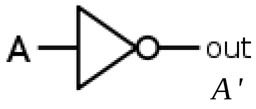
AND



OR



NOT (negation)

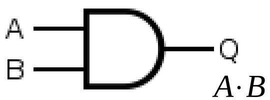


Gate images:
https://en.wikipedia.org/wiki/AND_gate , https://en.wikipedia.org/wiki/OR_gate
https://en.wikipedia.org/wiki/Inverter_(logic_gate)

Logic gates

Using truth tables, we can see the results of
A · B, A + B, and A’
for some given inputs.
Now we use 0 and 1 instead of “T” and “F”.

AND

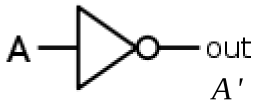


A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

OR

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

NOT (negation)

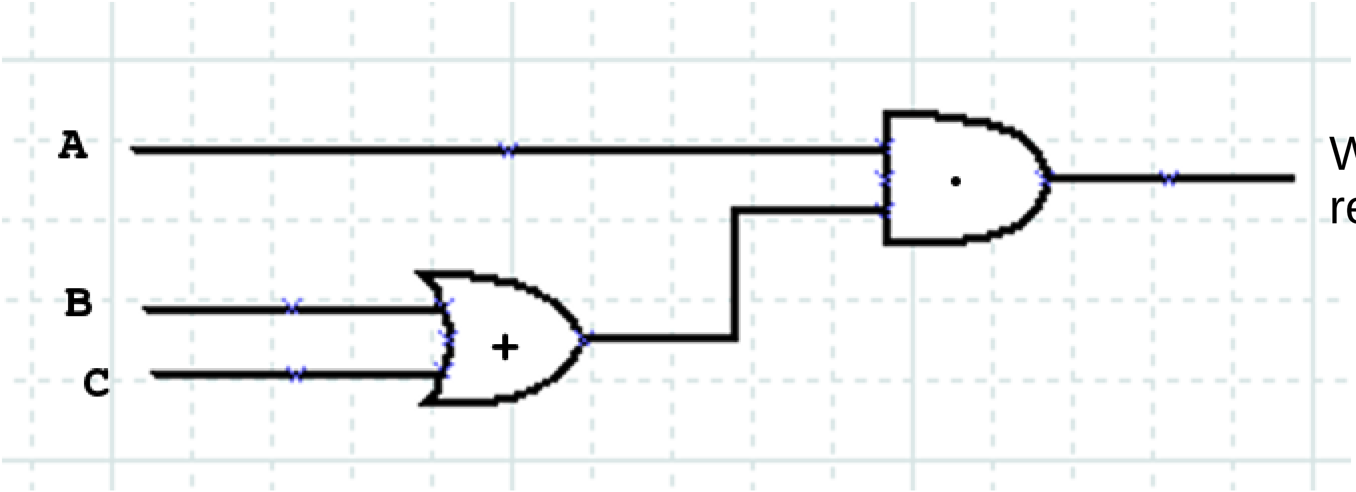


A	A’
0	1
1	0

Gate images:
https://en.wikipedia.org/wiki/AND_gate , https://en.wikipedia.org/wiki/OR_gate
https://en.wikipedia.org/wiki/Inverter_(logic_gate)

Logic gates

Example: Consider the following diagram. What is the <u>Boolean algebra</u> equation, and the truth table?

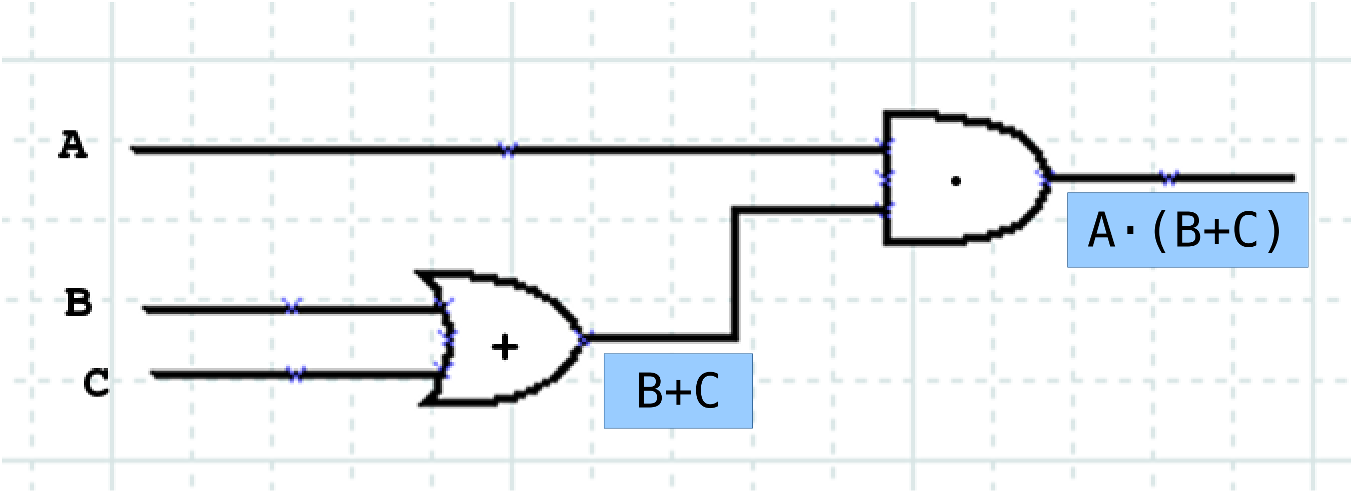


What does this final result symbolize?

Gate example from
http://www.cs.gordon.edu/local/courses/cps111/Notes/Boolean%20logic/boolgates.html

Logic gates

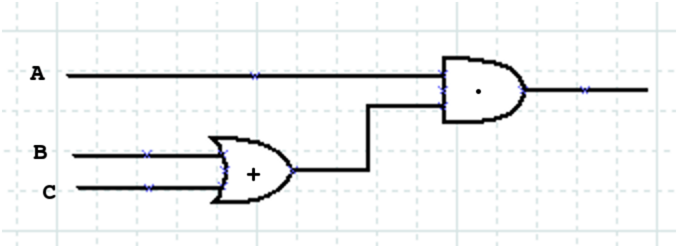
Example: Consider the following diagram. What is the <u>Boolean algebra</u> equation, and the truth table?



And what is the truth table for this diagram?

Logic gates

Example: Consider the following diagram. What is the <u>Boolean algebra</u> equation, and the truth table?



A	B	C	B+C	A·(B+C)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Boolean algebra properties

Just like with our logic and set properties, we also have similar properties for Boolean algebra:

(a) Commutative: $a \cdot b = b \cdot a$ $\quad a + b = b + a$

(b) Distributive: $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
$a + (b \cdot c) = (a + b) \cdot (a + c)$

(c) Identity: $a \cdot 1 = a$ $\quad a + 0 = a$

(d) Negation: $a + a' = 1$ $\quad a \cdot a' = 0$

Boolean algebra properties

Just like with our logic and set properties, we also have similar properties for Boolean algebra:

(a) Commutative: $a \cdot b = b \cdot a$ $a + b = b + a$

So it is possible for two gate diagrams to be **logically equivalent**, even if they're designed differently.

(b) D ... $(a \cdot b) + (a \cdot c)$

$a + ($ ...

Think of how this might affect the design of the physical hardware – we want the simplest design to save space and components.

(c) Identity: $a \cdot 1 = a$ $a + 0 = a$

(d) Negation: $a + a' = 1$ $a \cdot a' = 0$

Practice

Example 2 from the textbook
Show that

$$a + (b \cdot c) = (a + b) \cdot (a + c)$$

with a truth table…

Practice

Example 2 from the textbook
Show that

$$a + (b \cdot c) = (a + b) \cdot (a + c)$$

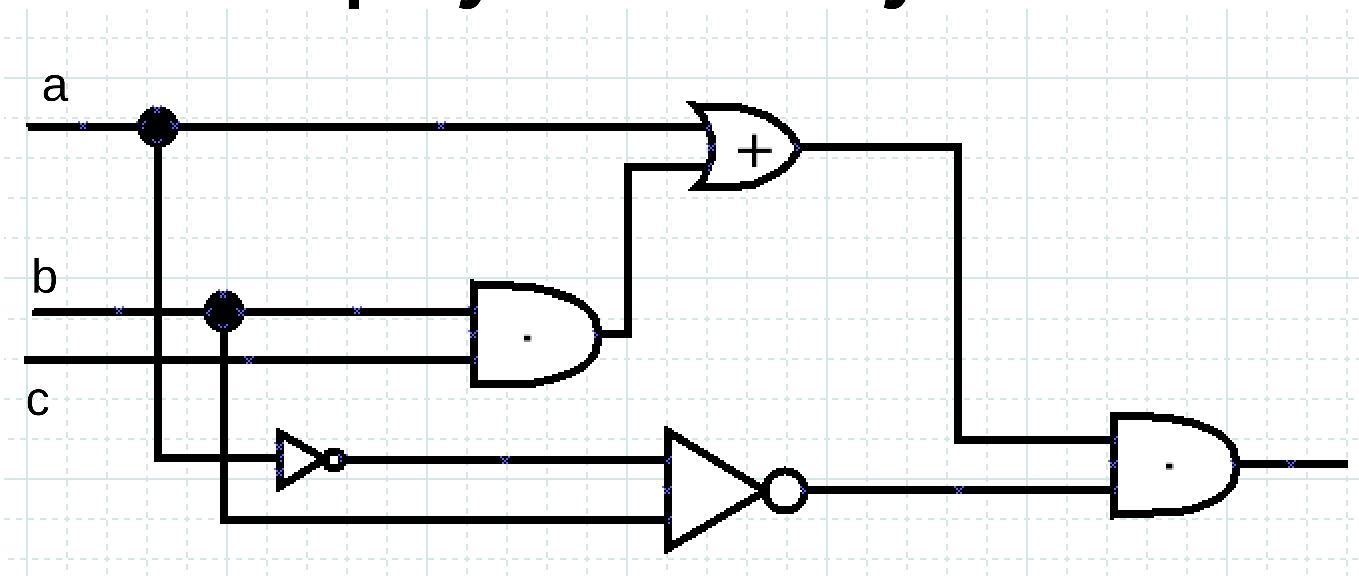
with a truth table...

a	b	c	b·c	a+b·c
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

a	b	c	a+b	a+c	(a+b)·(a+c)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

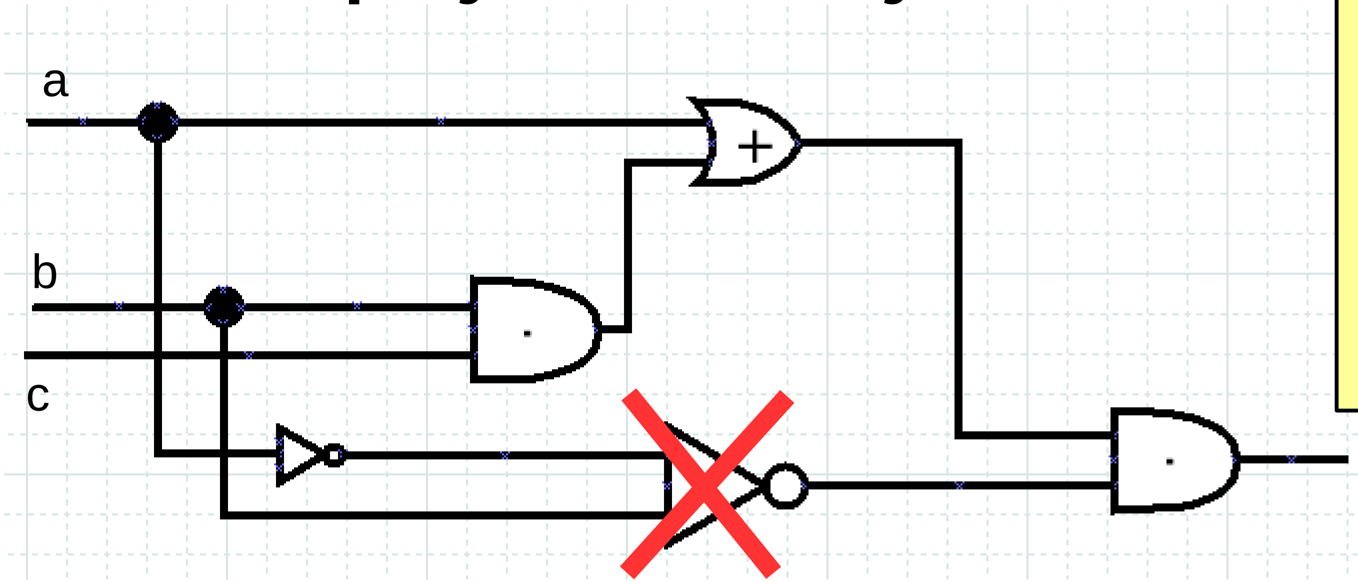
Practice

Practice problem 2 from the textbook
Write the Boolean expression for the circuit diagram, and simplify as much as you can.



Practice

Practice problem 2 from the textbook
Write the Boolean expression for the circuit diagram, and simplify as much as you can.




```
    (a + bc)(a'b)'
=   (a + bc)(a + b')
=   (a + (bc)b'
=   a + cbb'
=   a + c0
=   a + 0
=   a
```

Karnaugh Maps

K-maps: Two variables

Karnaugh maps are used to simplify boolean expressions. Instead of using a truth table, we draw our expression in a grid. We display all possible terms, and simplify.

	y	y’
x		
x’		

K-maps: Two variables

__Example:__ **Let's look at** `xy + xy'`

If we look at it, we can see that it says "x and y, or x and not y", which we can logically reduce on our own, but let's use a map.

K-maps: Two variables

<u>Example:</u> **Let’s look at** `xy + xy’`

First, we mark the two <u>products</u> on our grid.

	y	y’
x	✓	✓
x’		

K-maps: Two variables

<u>Example:</u> **Let's look at** `xy + xy'`

First, we mark the two <u>products</u> on our grid.

Then we draw a rectangle around adjacent blocks that have been ✓d off.

Rectangles are what represent where we can simplify.

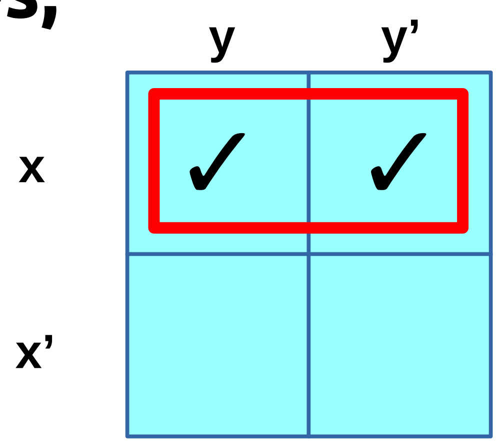
	y	y'
x	✓	✓
x'		

K-maps: Two variables

<u>Example:</u> **Let's look at** `xy + xy' = x`

The rectangle contains all states in the "x" row, so the simplified version here is just `x`**.**

If we had multiple rectangles, then our simplified form would be the sum of these rectangles.



K-maps: Two variables

<u>Example:</u> **Let's look at** $xy + xy' = x$

We can also verify this mathematically:

$$
\begin{aligned}
& xy + xy' \\
&= x(y + y') \qquad \textcolor{blue}{\textit{y + y' = 1}} \\
&= x \cdot 1 \\
&= x
\end{aligned}
$$

	y	y'
x	✓	✓
x'		

Practice

Example 6 from book: Simplify xy’ + x’y’ with a Karnaugh map.

	y	y’
x		
x’		

Practice

Example 6 from book: Simplify $xy + xy' + x'y$ with a Karnaugh map.

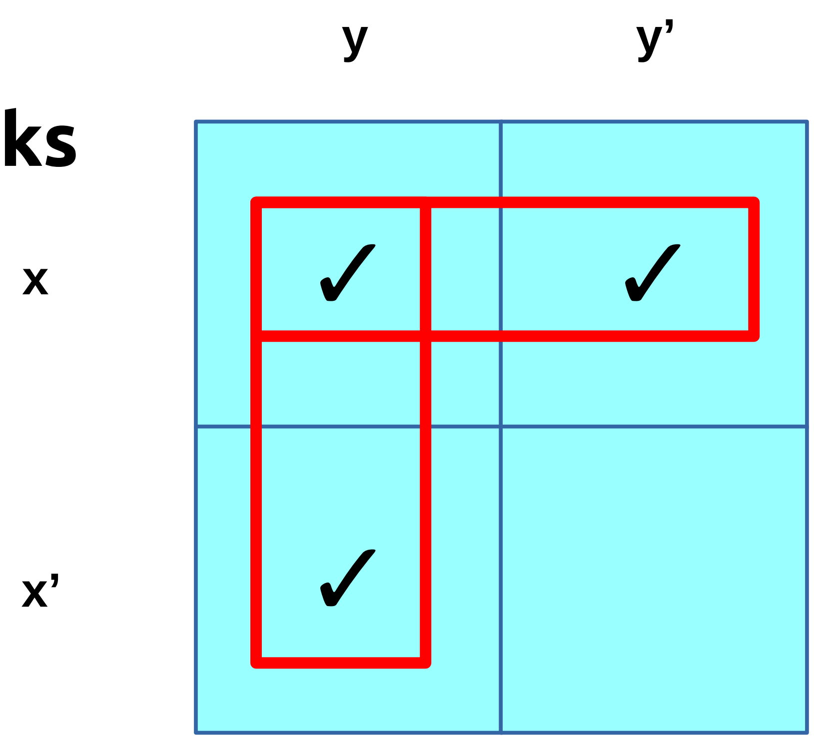
1. Place checkmarks

	y	y'
x	✓	✓
x'	✓	

Practice

Example 6 from book: Simplify `xy + xy' + x'y` **with a Karnaugh map.**

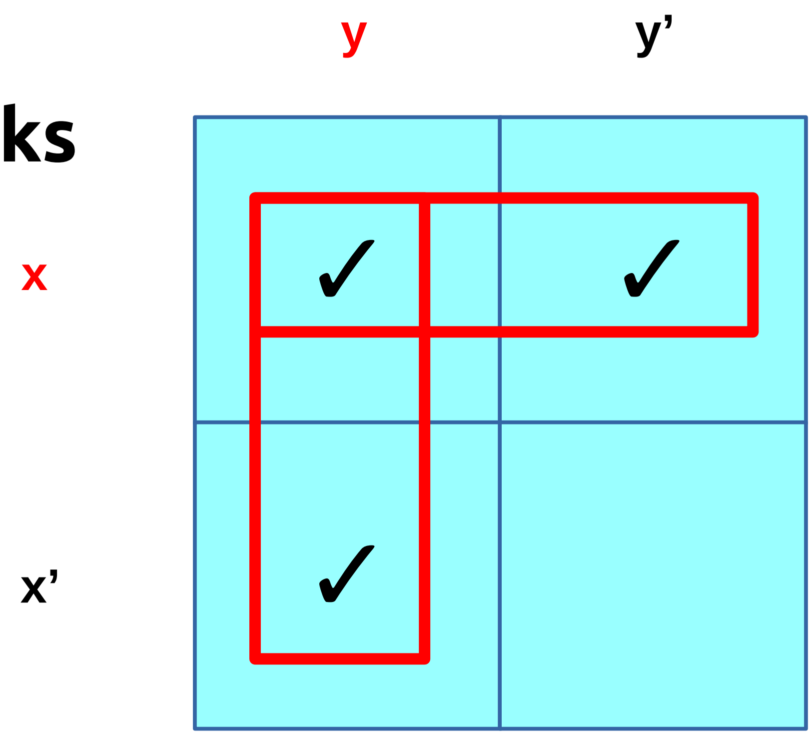
1. Place checkmarks
2. Region off adjacent checks



Practice

Example 6 from book: Simplify $xy + xy' + x'y$ with a Karnaugh map.

1. Place checkmarks
2. Region off adjacent checks
3. Result is the sum of these regions (based on row/col that is filled in)



$$x + y$$

No simplification

Sometimes when you build a map, there is just no simplification, such as with this grid.

There are no adjacent blocks with ✓ marks, so it cannot be simplified.

	y	y’
x		✓
x’	✓	

K-maps: Three variables

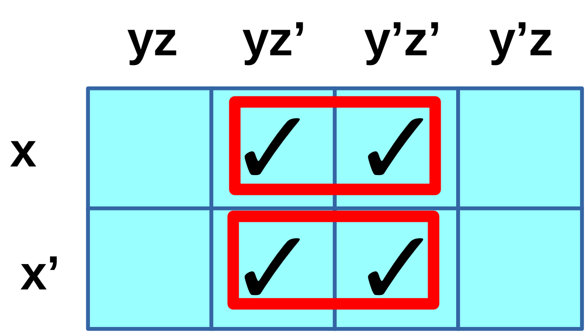
What about when we have three variables? Then we build our Karnaugh map grid as it is below.

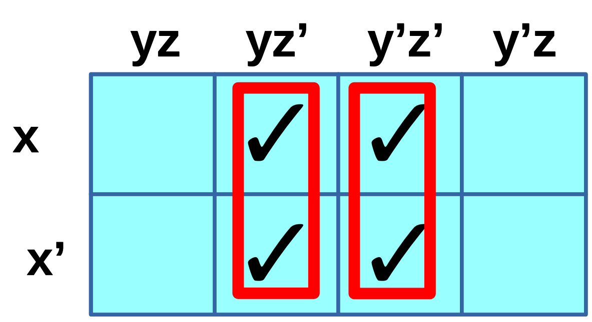
For the columns, labels in each column that are next to each other differ by <u>only one variable</u> – in other words, you can’t go from yz **to** y’z’ **in neighbors.**

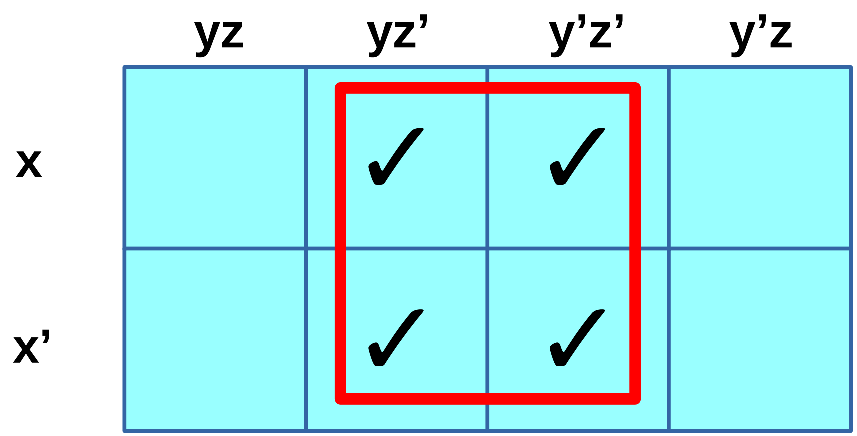
	yz	yz’	y’z’	y’z
x				
x’				

Simplification guidelines

There are also additional guidelines for us to find the simplest expression with a Karnaugh map. There are different ways to find rectangles, but they're not necessarily the most simplified forms.







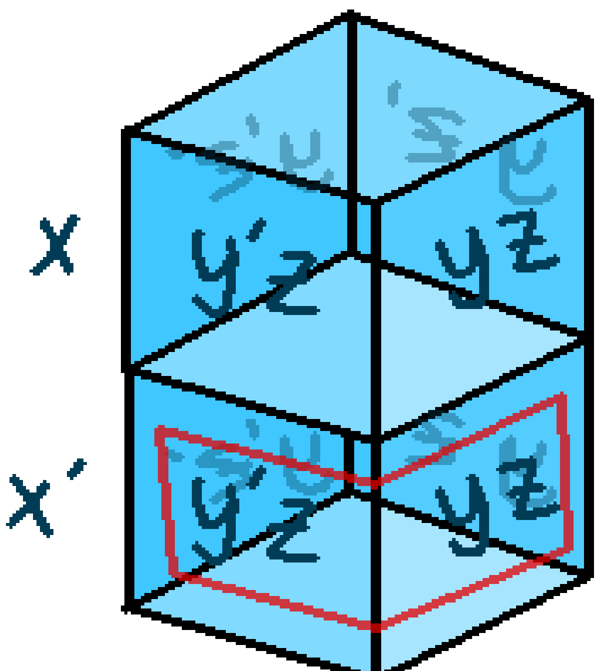
Wrapping around

For these maps, our rectangles can also WRAP AROUND.

	yz	yz’	y’z’	y’z
x				
x’	✓			✓

Wrapping around

yz and y'z are only one tiny difference apart, so perhaps it might be useful to think of the map as a 3D shape instead, which keeps looping.



	yz	yz'	y'z'	y'z
x				
x'	✓			✓

Smallest # of rectangles

Choose rectangles in a way that yields the smallest number of total rectangles, and that each rectangle is as large as possible.

	yz	yz’	y’z’	y’z
x		✓	✓	✓
x’		✓	✓	

Size 3 not allowed

Lengths of size 3 are <u>not allowed</u>: only length 1, 2, or 4.
(If it were size 3, you wouldn't have full coverage of a row. 1 is allowed because it would be anything left over.)

	yz	yz'	y'z'	y'z
x		✓	✓	✓
x'		✓	✓	

Missing variables

If the equation has a term without one of the variables, then realize that the expression is true for both “true” and “false” for that missing variable.

xy **means** xyz **and** xyz'**.**

	yz	yz’	y’z’	y’z
x	✓	✓		
x’				

Karnaugh map for *xy*

Examples

Examples

xyz + xyz’ + xy’z’ + xy’z = x

The full row for x is filled in with ✓ marks, meaning that x is the only variable that actually affects the outcome of the equation.

	yz	yz’	y’z’	y’z
x	✓	✓	✓	✓
x’				

Examples

Examples

```
xyz' + xy'z' + x'yz' + xy'z'
= z'
```

	yz	yz'	y'z'	y'z
x		✓	✓	
x'		✓	✓	

Notice that the two columns that are "full" are the ones that belong to z'.

z' is always present, whether we have x, x', y, or y'.

Practice

Example 9 from the book: Simplify
$xyz + xy'z' + xy'z + x'yz + x'y'z$
to the simplest form.

	yz	yz'	y'z'	y'z
x	✓		✓	✓
x'	✓			✓

Practice

Example 9 from the book: Simplify
`xyz + xy'z' + xy'z + x'yz + x'y'z`
to the simplest form.

Here's the map…

	yz	yz'	y'z'	y'z
x	✓		✓	✓
x'	✓			✓

Practice

Example 9 from the book: Simplify

`xyz + xy'z' + xy'z + x'yz + x'y'z`

to the simplest form.

Here's the map…

Result is

`z + xy'`

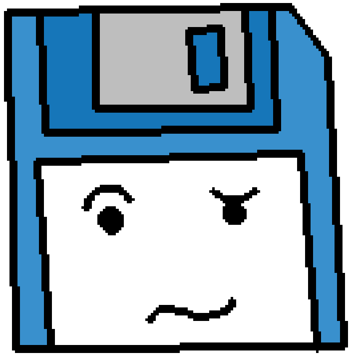
	yz	yz'	y'z'	y'z
x	✓		✓	✓
x'	✓			✓

K-maps: Four variables

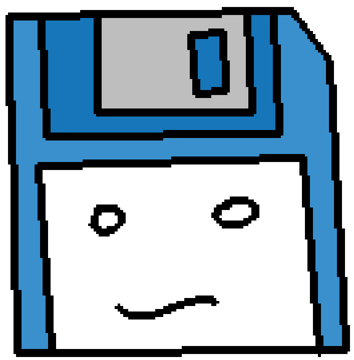
“Are we done yet?!”

Not yet… we can also work with maps of four variables.

“…dang.”



K-maps: Four variables



	zw	zw’	z’w’	z’w
xy				
xy’				
x’y’				
x’y				

K-maps: Four variables

Aren’t you exited?

Well, anyway…

	zw	zw’	z’w’	z’w
xy				
xy’				
x’y’				
x’y				

K-maps: Four variables

Keep in mind with these maps, now for columns and rows both, labels that are next to each other differ only by one.

And again, we can have wrap-around, but this time in both directions.

	zw	zw’	z’w’	z’w
xy				
xy’				
x’y’				
x’y				

K-maps: Four variables

Example 10 from the book: Simplify

$xyz'w' + xy'z'w' + xy'z'w + x'y'z'w' + x'yz'w'$

	zw	zw'	z'w'	z'w
xy				
xy'				
x'y'				
x'y				

K-maps: Four variables

Example 10 from the book: Simplify

xyz'w' + xy'z'w' + xy'z'w + x'y'z'w' + x'yz'w'

Map…

	zw	zw'	z'w'	z'w
xy			✓	
xy'			✓	✓
x'y'			✓	
x'y			✓	

K-maps: Four variables

Example 10 from the book: Simplify

xyz’w’ + xy’z’w’ + xy’z’w + x’y’z’w’ + x’yz’w’

Map…

Regions…

	zw	zw’	z’w’	z’w
xy			✓	
xy’			✓	✓
x’y’			✓	
x’y			✓	

K-maps: Four variables

Example 10 from the book: Simplify

xyz’w’ + xy’z’w’ + xy’z’w + x’y’z’w’ + x’yz’w’

Map…

Regions…

= xy’z’ +

	zw	zw’	z’w’	z’w
xy			✓	
xy’			✓	✓
x’y’			✓	
x’y			✓	

K-maps: Four variables

Example 10 from the book: Simplify

`xyz'w' + xy'z'w' + xy'z'w + x'y'z'w' + x'yz'w'`

Map…

Regions…

`= xy'z' + z'w'`

	zw	zw'	z'w'	z'w
xy			✓	
xy'			✓	✓
x'y'			✓	
x'y			✓	

K-maps: Four variables

Example 10 from the book: Simplify

xyz’w’ + xy’z’w’ + xy’z’w + x’y’z’w’ + x’yz’w’

Map…

Regions…

= xy’z’ + z’w’

Got it?

	zw	zw’	z’w’	z’w
xy			✓	
xy’			✓	✓
x’y’			✓	
x’y			✓	

Practice

Example 12 from the book: Build the simplest regions for this map.

	zw	zw’	z’w’	z’w
xy	✓			✓
xy’	✓	✓		
x’y’				
x’y	✓			✓

Practice

Example 12 from the book: Build the simplest regions for this map.

Don’t forget about being able to wrap around in all directions!

	zw	zw’	z’w’	z’w
xy	✓			✓
xy’	✓	✓		
x’y’				
x’y	✓			✓

Practice

Example 13 from the book: Simplify

$$y'w' + yz + x'yw'$$

	zw	zw’	z’w’	z’w
xy				
xy’				
x’y’				
x’y				

Practice

Example 13 from the book: Simplify

y'w' + yz + x'yw'

Remember that when variables are missing, we add them in as both the normal and prime forms.

y'w' =
(xz)y'w' **or** (x'z)y'w'
or (x'z')y'w' **or** (xz')y'w'...

	zw	zw'	z'w'	z'w
xy	✓	✓		
xy'		✓	✓	
x'y'		✓	✓	
x'y	✓	✓	✓	

Practice

Example 13 from the book: Simplify

$$y'w' + yz + x'yw'$$

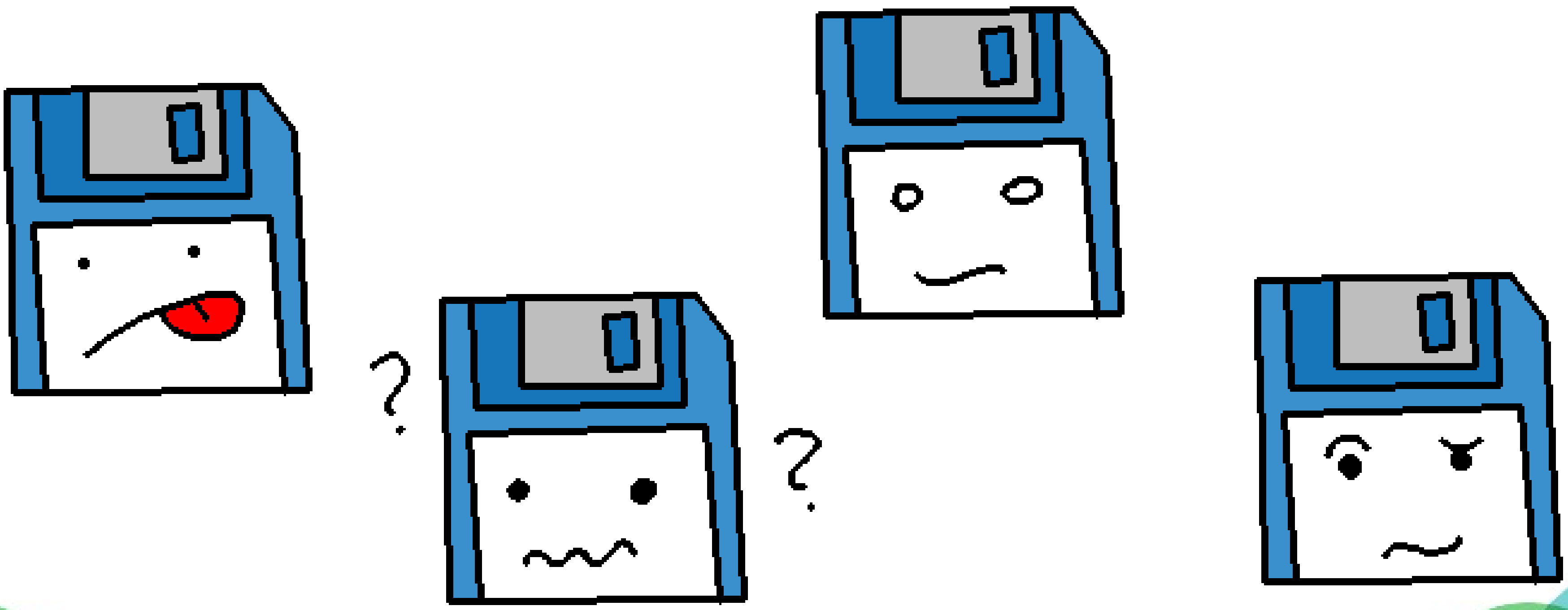
So the simplest form is

$$y'w' + yz + x'w'$$

(tried to use circles to make it easier to see the overlapped regions)

	zw	zw'	z'w'	z'w
xy	✓	✓		
xy'		✓	✓	
x'y'		✓	✓	
x'y	✓	✓	✓	

Alright, we're done now.



	y	y’
x		✓
x’	✓	✓

	yz	yz’	y’z’	y’z
x				
x’				