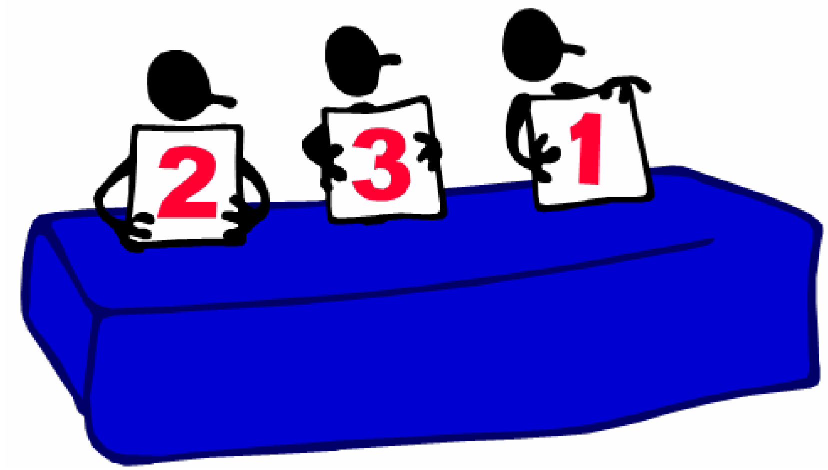
Boolean algebra



- Yesterday we talked about how analog voltages can represent the logical values true and false.
  - We introduced the basic Boolean operations AND, OR and NOT, which can be implemented in hardware with primitive logic gates.
  - It follows that any Boolean expression, composed of basic operations, can be computed with a circuit of primitive gates.
- Today we'll present the axioms of Boolean algebra, and discuss how they help us simplify functions and circuits.

Operations and gates review

Operation:	AND (product) of two inputs	OR (sum) of two inputs	NOT (complement) of one input
Expression:	xy or $x \bullet y$	$x + y$	x' or $\overline{x}$

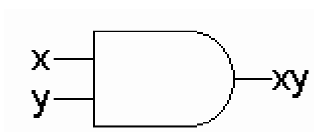
Truth table:

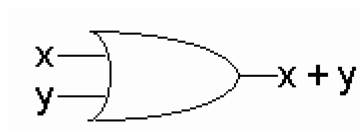
x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

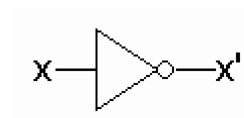
x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

x	x’
0	1
1	0

Logic gate symbol:

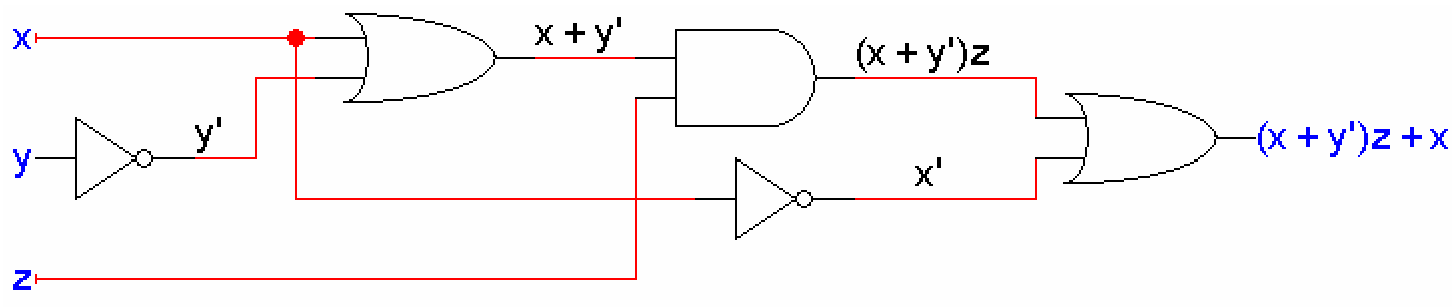






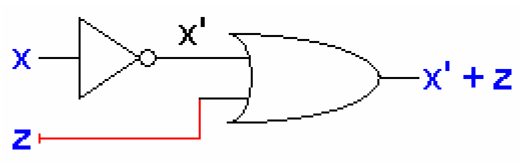
Expressions and circuits

- We can build a circuit for any Boolean expression by connecting primitive logic gates in the correct order.
- Yesterday we showed the example circuit below, which accepts inputs x, y and z, and produces the output (x + y')z + x'.



Simplifying circuits

- The big circuit on the last page is actually *equivalent* to this simpler one.



- Simpler hardware is almost always better.
  - In many cases, simpler circuits are faster.
  - Less hardware means lower costs.
  - A smaller circuit also consumes less power.
- So how were we able to simplify this particular circuit?

Smaller is better.

The definition of a Boolean algebra

- The secret is Boolean algebra, which lets us simplify Boolean functions just as regular algebra allows us to manipulate arithmetic functions.
- A Boolean algebra requires:
  - A set of values with at least two elements, denoted 0 and 1
  - Two binary (two-argument) operations + and •
  - A unary (one-argument) operation ’
- These values and operations must satisfy the axioms shown below.

$x + 0 = x$	$x \bullet 1 = x$	
$x + 1 = 1$	$x \bullet 0 = 0$	
$x + x = x$	$x \bullet x = x$	
$x + x' = 1$	$x \bullet x' = 0$	
$(x')' = x$		
$x + y = y + x$	$xy = yx$	Commutative
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associative
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	Distributive
$(x + y)' = x'y'$	$(xy)' = x' + y'$	DeMorgan’s Law

Satisfying the axioms

- Fortunately, the AND, OR and NOT operations that we defined do satisfy all of the axioms.

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

- For example, we can show that the axiom $x + x' = 1$ always holds.
  - There are only two possible values for x, 0 or 1.
  - The complement of these values is 1 and 0, by our definition of NOT.
  - According to our definition of OR, $0 + 1 = 1$, and $1 + 0 = 1$.

x	x'	$x + x'$
0	1	1
1	0	1

Similarities with regular algebra

- The axioms in blue look just like regular algebraic rules—this is one of the reasons we overload the + and • symbols for Boolean operations.
- The associative laws show that there is no ambiguity in an expression like xyz or $x + y + z$, so we can use multi-input primitive gates as well as our original two-input gates.

$x + 0 = x$	$x \bullet 1 = x$	
$x + 1 = 1$	$x \bullet 0 = 0$	
$x + x = x$	$x \bullet x = x$	
$x + x' = 1$	$x \bullet x' = 0$	
$(x')' = x$		
$x + y = y + x$	$xy = yx$	Commutative
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associative
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	Distributive
$(x + y)' = x'y'$	$(xy)' = x' + y'$	DeMorgan's Law

The complement operation

- The magenta axioms deal with the complement operator.
- The first three make sense if you think about some English examples.
  - "It is snowing or it is not snowing" is always true $(x + x' = 1)$
  - "It is snowing and it is not snowing" can never be true $(x \bullet x' = 0)$
  - "I am not not handsome" means "I am handsome" $((x')' = x)$

$x + 0 = x$	$x \bullet 1 = x$	
$x + 1 = 1$	$x \bullet 0 = 0$	
$x + x = x$	$x \bullet x = x$	
$x + x' = 1$	$x \bullet x' = 0$	
$(x')' = x$		
$x + y = y + x$	$xy = yx$	Commutative
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associative
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	Distributive
$(x + y)' = x'y'$	$(xy)' = x' + y'$	DeMorgan's Law

DeMorgan’s Laws

- DeMorgan’s Laws explain how to complement arbitrary expressions.

$$(x + y)' = x'y' \qquad\qquad (xy)' = x' + y'$$

- Here are some examples in English.
  - “I’m not rich-or-famous” means that I’m not rich *and* I’m not famous.
  - “I am not old-and-bald” means “I am not old *or* I am not bald.” But I could be (1) young and bald, (2) young and hairy, or (3) old and hairy.

Other differences from regular algebra

- Finally, the red axioms are completely different from regular algebra.
- The first three make sense logically.
  - "Anything or true" always holds, even if "anything" is false ($x + 1 = 1$)
  - "I am handsome or I am handsome" is redundant ($x + x = x$)
  - "I am handsome and I am handsome" is also redundant ($x \bullet x = x$)
- The last one, $x + yz = (x + y)(x + z)$, is the least intuitive, but you can prove it using truth tables or the other axioms.

$x + 0 = x$	$x \bullet 1 = x$	
$x + 1 = 1$	$x \bullet 0 = 0$	
$x + x = x$	$x \bullet x = x$	
$x + x' = 1$	$x \bullet x' = 0$	
$(x')' = x$		
$x + y = y + x$	$xy = yx$	Commutative
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associative
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	Distributive
$(x + y)' = x'y'$	$(xy)' = x' + y'$	DeMorgan's Law

Simplifications

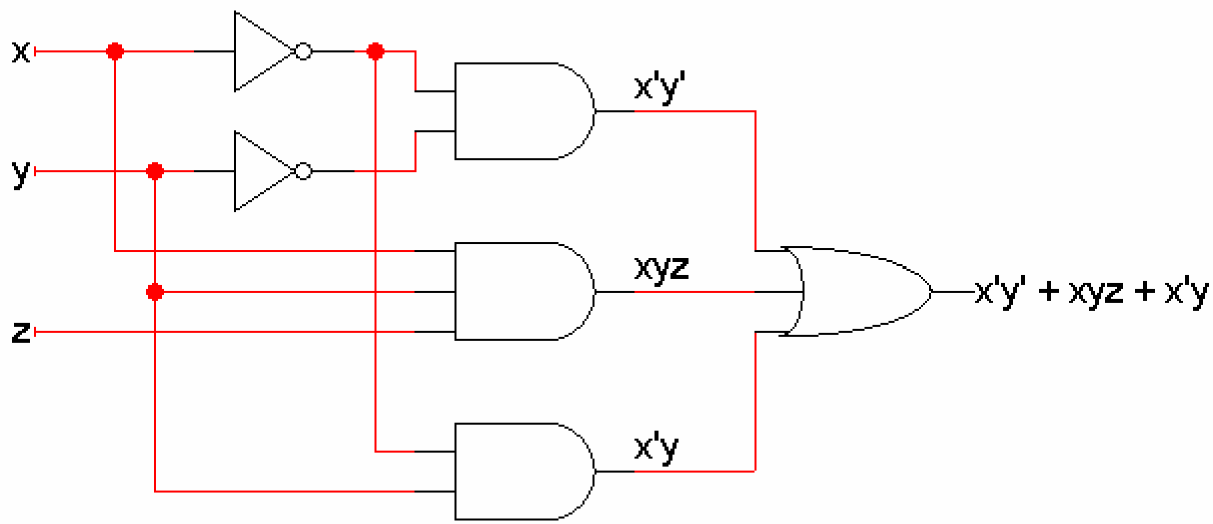
- Now we can use these axioms to simplify expressions and circuits.

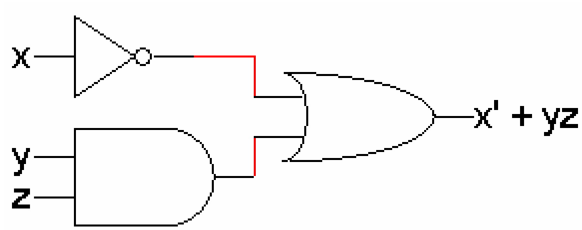
$x'y' + xyz + x'y$	$= x'y' + x'y + xyz$	[Commutative]
	$= x'(y' + y) + xyz$	[Distributive]
	$= (x' \bullet 1) + xyz$	[$y' + y = 1$]
	$= x' + xyz$	[$x' \bullet 1 = x'$]
	$= (x' + x)(x' + yz)$	[Distributive!]
	$= 1 \bullet (x' + yz)$	[$x' + x = 1$]
	$= x' + yz$	

$x + 0 = x$	$x \bullet 1 = x$	
$x + 1 = 1$	$x \bullet 0 = 0$	
$x + x = x$	$x \bullet x = x$	
$x + x' = 1$	$x \bullet x' = 0$	
$(x')' = x$		
$x + y = y + x$	$xy = yx$	Commutative
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associative
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	Distributive
$(x + y)' = x'y'$	$(xy)' = x' + y'$	DeMorgan's Law

Simpler expressions yield simpler hardware

- Here are circuits corresponding to the original and simplified expressions.





Proofs with truth tables

- We also can prove that two expressions are equivalent by showing that they always produce the same results for the same inputs.

x	y	x + y	(x + y)'
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

x	y	x'	y'	x'y'
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

- Here are truth tables proving one of DeMorgan's Laws, (x + y)' = x'y'.
  - The leftmost columns in each table show all the possible inputs.
  - The columns on the right are the outputs.
  - Additional columns can aid in showing intermediate results.
- Both of the output columns are the same, so we know that (x + y)' and x'y' must be equivalent.

Duality

- There's a reason why the table of axioms has two columns. The laws on the left and right are duals of each other.
  - The AND and OR operators are exchanged.
  - The constant values 0 and 1 are also exchanged.
- The dual of *any* equation is always true. If E and F are two equivalent expressions, the dual of E will also be equivalent to the dual of F.

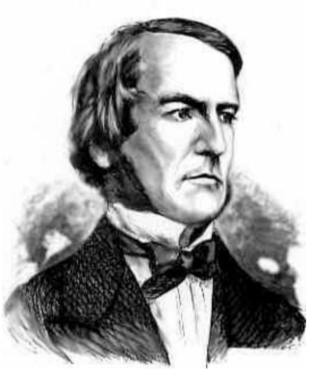
$x + 0 = x$	$x \bullet 1 = x$	
$x + 1 = 1$	$x \bullet 0 = 0$	
$x + x = x$	$x \bullet x = x$	
$x + x' = 1$	$x \bullet x' = 0$	
$(x')' = x$		
$x + y = y + x$	$xy = yx$	Commutative
$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$	Associative
$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$	Distributive
$(x + y)' = x'y'$	$(xy)' = x' + y'$	DeMorgan's Law

Some more laws

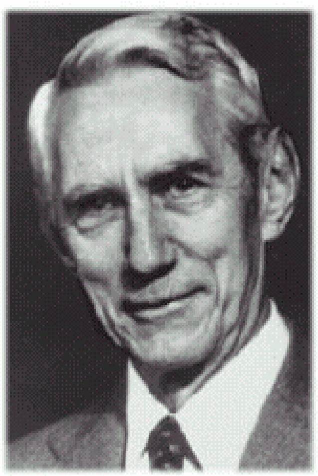
- Some other useful equations are shown below.
  - They can all be proven from the axioms we already showed.
  - Notice that each law also has a dual.
- Feel free to use these in homeworks and exams.

$x + xy = x$	$x(x + y) = x$
$xy + xy' = x$	$(x + y)(x + y') = x$
$x + x'y = x + y$	$x(x' + y) = xy$
$xy + x'z + yz = xy + x'z$	$(x + y)(x' + z)(y + z) = (x + y)(x' + z)$

Why is it called Boolean algebra?



- It was invented by George Boole way back in the 1850s!
- Obviously, that was before they had digital cameras.



- It wasn't until about 1937 that Claude Shannon got the idea to apply Boolean algebra to circuit design.
- This, as well as several other things, made Shannon so rich-and-famous that he retired when he was just 50.

Complementing a truth table

- The complement of a function should output 0 when the original function outputs 1, and vice versa.
- In a truth table, we can just exchange 0 and 1 in the output column.
  - On the left is a truth table for $f(x,y,z) = (x + y')z + x'$.
  - On the right is the table for the complement of f, denoted $f'(x,y,z)$.

x	y	z	f(x,y,z)
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

x	y	z	f'(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Complementing an expression

- To complement an expression, you can use DeMorgan's Laws to keep "pushing" the NOT operator inwards, all the way to the literals.

$$f(x,y,z) = (x + y')z + x'$$

$$\begin{aligned}
f'(x,y,z) &= ((x + y')z + x')' && \text{[complementing both sides]} \\
&= ((x + y')z)' \bullet (x')' && \text{[because } (x + y)' = x'y' \text{]} \\
&= ((x + y')' + z') \bullet x && \text{[} (xy)' = x' + y' \text{, and } (x')' = x \text{]} \\
&= (x'y + z') \bullet x && \text{[} (x + y)' = x'y' \text{ again]}
\end{aligned}$$

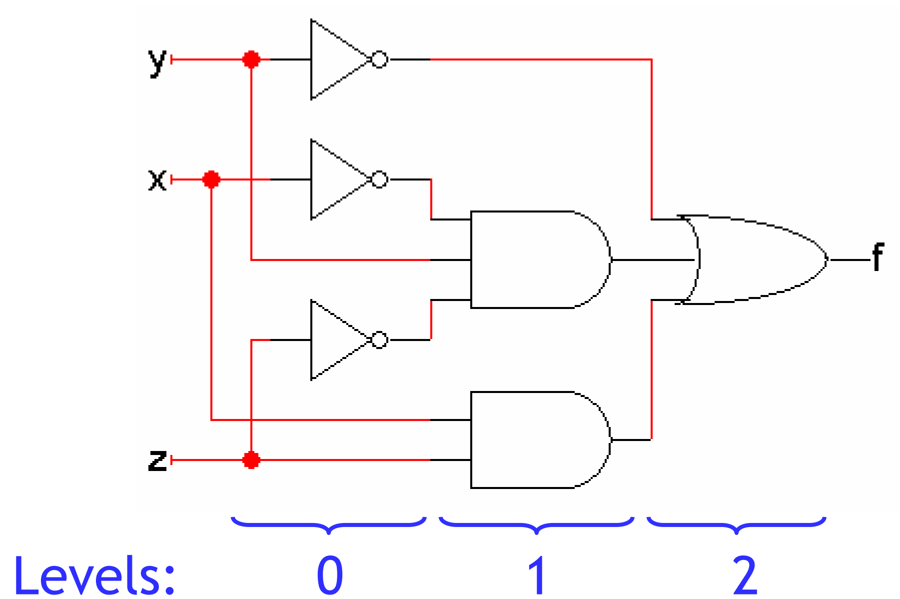
- Another clever method of complementing an expression is to take the dual of the expression, and then complement each literal.
  - The dual of $(x + y')z + x'$ is $(xy' + z) \bullet x'$.
  - Complementing each literal yields $(x'y + z') \bullet x$.
  - So $f'(x,y,z) = (x'y + z') \bullet x$.

Sum of products expressions

- There are many equivalent ways to write a function, but some forms turn out to be more useful than others.
- A sum of products or SOP expression consists of:
  – One or more terms *summed* (OR'ed) together.
  – Each of those terms is a *product of literals*.

$$f(x, y, z) = y' + x'yz' + xz$$

- Sum of products expressions can be implemented with two-level circuits.



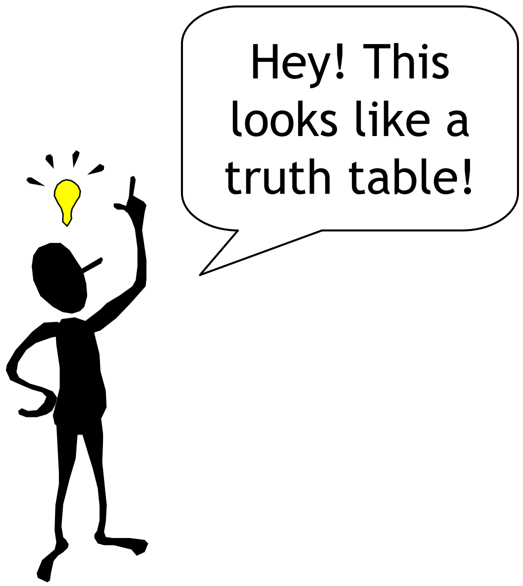
Minterms

- A minterm is a special product of literals, in which each input variable appears exactly once.
- A function with n input variables has 2^n possible minterms.
- For instance, a three-variable function f(x,y,z) has 8 possible minterms:

x'y'z'	x'y'z	x'y z'	x'y z
x y'z'	x y'z	x y z'	x y z

- Each minterm is true for exactly one combination of inputs.

Minterm	True when	Shorthand
x'y'z'	xyz = 000	m_0
x'y'z	xyz = 001	m_1
x'y z'	xyz = 010	m_2
x'y z	xyz = 011	m_3
x y'z'	xyz = 100	m_4
x y'z	xyz = 101	m_5
x y z'	xyz = 110	m_6
x y z	xyz = 111	m_7



Sum of minterms expressions

- A sum of minterms is a special kind of sum of products.
- Every function can be written as a *unique* sum of minterms expression.
- A truth table for a function can be rewritten as a sum of minterms just by finding the table rows where the function output is 1.

x	y	z	C(x,y,z)	C'(x,y,z)
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

$$
\begin{aligned}
C &= x'yz + xy'z + xyz' + xyz \\
&= m_3 + m_5 + m_6 + m_7 \\
&= \Sigma m(3,5,6,7)
\end{aligned}
$$

$$
\begin{aligned}
C' &= x'y'z' + x'y'z + x'yz' + xy'z' \\
&= m_0 + m_1 + m_2 + m_4 \\
&= \Sigma m(0,1,2,4)
\end{aligned}
$$

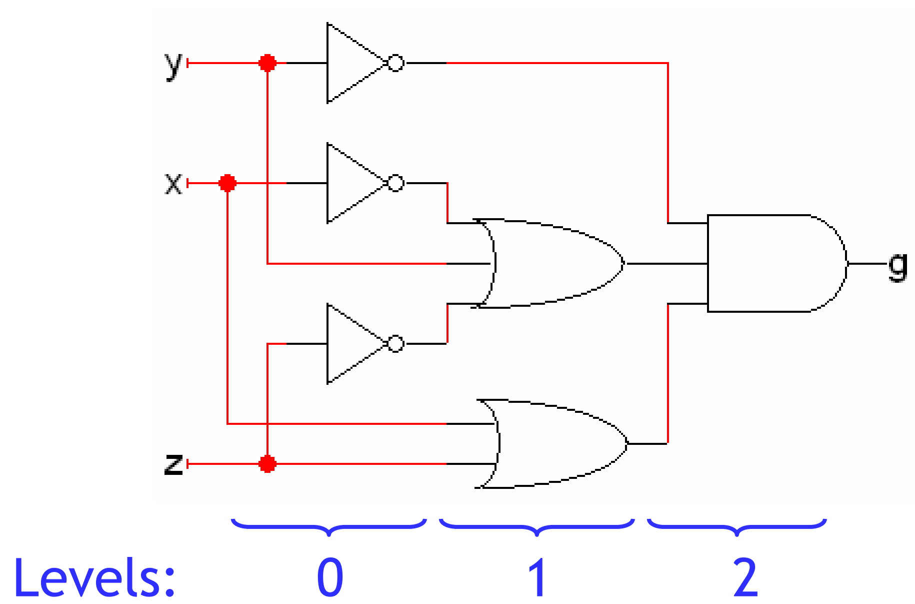
C' contains all the minterms *not* in C, and vice versa.

Product of sums expressions

- As you might expect, we can work with the duals of these ideas too.
- A product of sums or POS consists of:
  - One or more terms *multiplied* (AND'ed) together.
  - Each of those terms is a *sum of literals*.

$$g(x, y, z) = y'(x' + y + z')(x + z)$$

- Products of sums can also be implemented with two-level circuits.



Maxterms

- A maxterm is a *sum* of literals where each input variable appears once.
- A function with *n* input variables has 2^n possible maxterms.
- For instance, a function with three variables x, y and z has 8 possible maxterms:

$x + y + z$	$x + y + z'$	$x + y' + z$	$x + y' + z'$
$x' + y + z$	$x' + y + z'$	$x' + y' + z$	$x' + y' + z'$

- Each maxterm is *false* for exactly one combination of inputs.

Maxterm	False when	Shorthand
$x + y + z$	$xyz = 000$	M_0
$x + y + z'$	$xyz = 001$	M_1
$x + y' + z$	$xyz = 010$	M_2
$x + y' + z'$	$xyz = 011$	M_3
$x' + y + z$	$xyz = 100$	M_4
$x' + y + z'$	$xyz = 101$	M_5
$x' + y' + z$	$xyz = 110$	M_6
$x' + y' + z'$	$xyz = 111$	M_7

Product of maxterms expressions

- Every function can also be written as a unique product of maxterms.
- A truth table for a function can be rewritten as a product of maxterms just by finding the table rows where the function output is 0.

x	y	z	C(x,y,z)	C'(x,y,z)
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

$$\begin{aligned} C &= (x + y + z)(x + y + z') \\ &\quad (x + y' + z)(x' + y + z) \\ &= M_0\, M_1\, M_2\, M_4 \\ &= \prod M(0,1,2,4) \end{aligned}$$

$$\begin{aligned} C' &= (x + y' + z')(x' + y + z') \\ &\quad (x' + y' + z)(x' + y' + z') \\ &= M_3\, M_5\, M_6\, M_7 \\ &= \prod M(3,5,6,7) \end{aligned}$$

C' contains all the maxterms *not* in C, and vice versa.

Minterms and maxterms, oh my!

- Now we've seen two different ways to write the function C, as a sum of minterms $\Sigma m(3,5,6,7)$ and as a product of maxterms $\Pi M(0,1,2,4)$.
- Notice the product term includes maxterm numbers whose corresponding minterms do not appear in the sum expression.

x	y	z	C(x,y,z)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$\begin{aligned} C &= x'yz + xy'z + xyz' + xyz \\ &= m_3 + m_5 + m_6 + m_7 \\ &= \Sigma m(3,5,6,7) \end{aligned}$$

$$\begin{aligned} C &= (x + y + z)(x + y + z') \\ &\quad (x + y' + z)(x' + y + z) \\ &= M_0\, M_1\, M_2\, M_4 \\ &= \prod M(0,1,2,4) \end{aligned}$$

The relationship revealed

- Every minterm m_i is the *complement* of its corresponding maxterm M_i.

Minterm	True when
(m_0) $x'y'z'$	xyz = 000
(m_1) $x'y'z$	xyz = 001
(m_2) $x'y\,z'$	xyz = 010
(m_3) $x'y\,z$	xyz = 011
(m_4) $x\,y'z'$	xyz = 100
(m_5) $x\,y'z$	xyz = 101
(m_6) $x\,y\,z'$	xyz = 110
(m_7) $x\,y\,z$	xyz = 111

Maxterm	False when
(M_0) $x + y + z$	xyz = 000
(M_1) $x + y + z'$	xyz = 001
(M_2) $x + y' + z$	xyz = 010
(M_3) $x + y' + z'$	xyz = 011
(M_4) $x' + y + z$	xyz = 100
(M_5) $x' + y + z'$	xyz = 101
(M_6) $x' + y' + z$	xyz = 110
(M_7) $x' + y' + z'$	xyz = 111

- For example, $m_4' = M_4$ because $(xy'z')' = x' + y + z$.

Converting between standard forms

- We can convert sums of minterms to products of maxterms algebraically.

$$
\begin{array}{lll}
C & = \Sigma m(3,5,6,7) & \\
 & & \\
C' & = \Sigma m(0,1,2,4) & \text{[C' contains the minterms not in C]} \\
 & = m_0 + m_1 + m_2 + m_4 & \\
 & & \\
(C')' & = (m_0 + m_1 + m_2 + m_4)' & \text{[complementing both sides]} \\
C & = m_0' \, m_1' \, m_2' \, m_4' & \text{[DeMorgan's law]} \\
 & = M_0 \, M_1 \, M_2 \, M_4 & \text{[from the previous page]} \\
 & = \prod M(0,1,2,4) &
\end{array}
$$

- The easy way is to replace minterms with maxterms, using the maxterm numbers that do not appear in the sum of minterms.

$$
\begin{aligned}
C &= \Sigma m(3,5,6,7) \\
&= \prod M(0,1,2,4)
\end{aligned}
$$

Summary

- We saw two ways to prove the equivalence of expressions.
  - Truth tables show that all possible inputs yield the same outputs.
  - Boolean algebra is especially useful for simplifying expressions, and therefore circuits as well.
- Expressions can be written in many ways, so standard representations like sums of products and sums of minterms are often useful. We will sometimes see products of sums and products of maxterms too.
- Tomorrow we'll introduce a more "graphical" simplification technique. Then we can start to build and analyze larger, more realistic circuits!