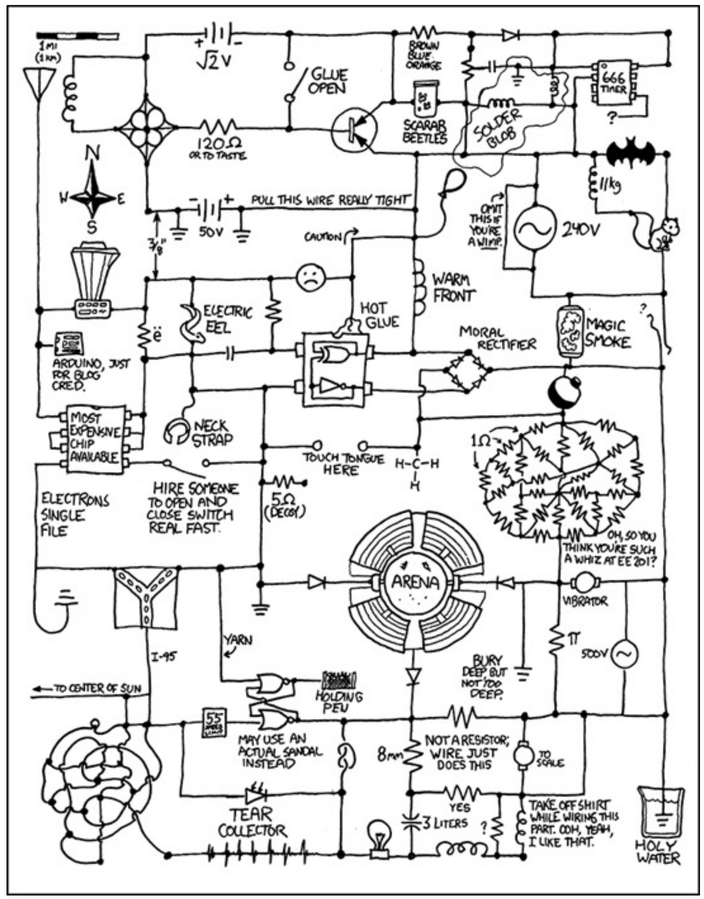
CSE 311: Foundations of Computing

Lecture 5: DNF, CNF and Predicate Logic



Administrivia

HW1 due tonight

HW2 posted tomorrow

- some tools available for testing equivalence chains
  - one is http://homes.cs.washington.edu/~kevinz/equiv-test/
  - another mentioned in the HW
- both are optional
  - also “beta” software

Last Time: Building Circuits

“Turn the Crank” Process:

1. write down a table showing desired 0/1 outputs
2. construct a Boolean algebra expression
   – term for each 1 in the column
   – sum (or) them to get all 1s
3. simplify the expression using equivalences
4. translate Boolean algebra to a circuit

(Since it’s “turn the crank”, software can do this for you.)

Warm-up Exercise

- Create a Boolean Algebra expression for "c" below in terms of the variables a and b

a	b	c
1	1	0
1	0	1
0	1	1
0	0	0

$$c = ab' + a'b$$

Warm-up Exercise

- Create a Boolean Algebra expression for "$\boldsymbol{c}$" below in terms of the variables $\boldsymbol{a}$ and $\boldsymbol{b}$

$$c = ab' + a'b$$

- Draw this as a circuit (using AND, OR, NOT)

1-bit Binary Adder

A	$0 + 0 = 0$ (with $C_{OUT} = 0$)
+ B	$0 + 1 = 1$ (with $C_{OUT} = 0$)
S	$1 + 0 = 1$ (with $C_{OUT} = 0$)
(C_{OUT})	$1 + 1 = 0$ (with $C_{OUT} = 1$)

1-bit Binary Adder

A	$0 + 0 = 0$ (with $C_{OUT} = 0$)
+ B	$0 + 1 = 1$ (with $C_{OUT} = 0$)
S	$1 + 0 = 1$ (with $C_{OUT} = 0$)
(C_{OUT})	$1 + 1 = 0$ (with $C_{OUT} = 1$)

Idea: chain these together to add larger numbers

Recall from elementary school:

```
  2 4 8
+ 3 7 5
```

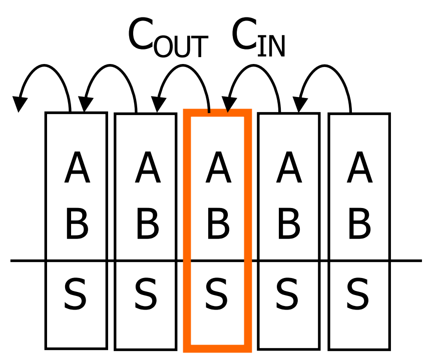
1-bit Binary Adder

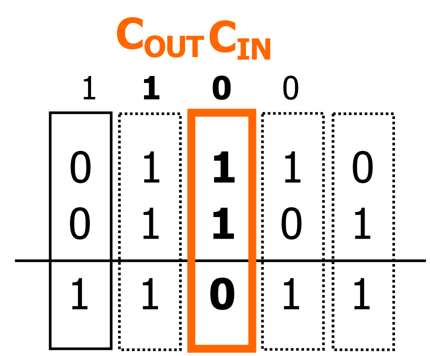
A
+ B
S
(C_{OUT})

$0 + 0 = 0$ (with $C_{OUT} = 0$)
$0 + 1 = 1$ (with $C_{OUT} = 0$)
$1 + 0 = 1$ (with $C_{OUT} = 0$)
$1 + 1 = 0$ (with $C_{OUT} = 1$)

Idea: These are chained together with a carry-in

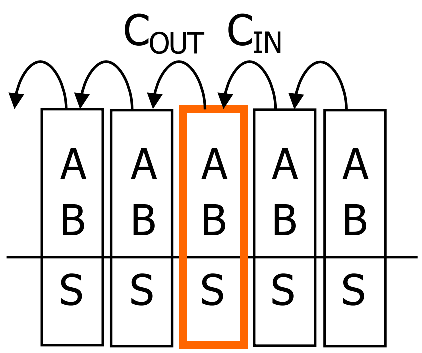
(C_{IN})
A
+ B
S
(C_{OUT})





1-bit Binary Adder

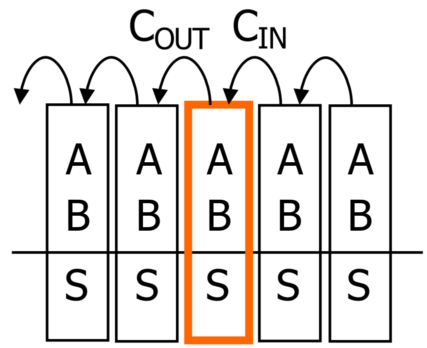
- **Inputs:** A, B, Carry-in
- **Outputs:** Sum, Carry-out



A	B	C_{IN}	C_{OUT}	S
0	0	0	0	0
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	1

1-bit Binary Adder

- **Inputs:** A, B, Carry-in
- **Outputs:** Sum, Carry-out



A	B	C_{IN}	C_{OUT}	S
0	0	0	0	0
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	1

Derive an expression for S

$A' \cdot B' \cdot C_{IN}$

$A' \cdot B \cdot C_{IN}'$

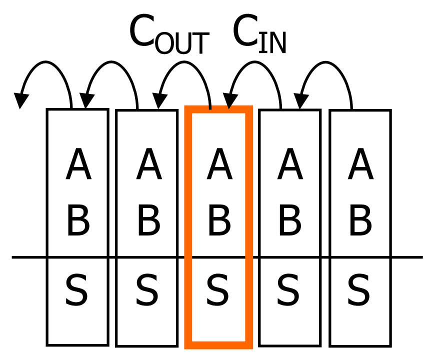
$A \cdot B' \cdot C_{IN}'$

$A \cdot B \cdot C_{IN}$

$$S = A' \cdot B' \cdot C_{IN} + A' \cdot B \cdot C_{IN}' + A \cdot B' \cdot C_{IN}' + A \cdot B \cdot C_{IN}$$

1-bit Binary Adder

- **Inputs:** A, B, Carry-in
- **Outputs:** Sum, Carry-out



A	B	C_{IN}	C_{OUT}	S
0	0	0	0	0
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	1

Derive an expression for C_{OUT}

$A' \cdot B \cdot C_{IN}$

$A \cdot B' \cdot C_{IN}$

$A \cdot B \cdot C_{IN}'$

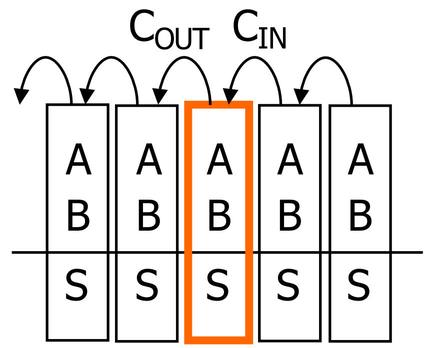
$A \cdot B \cdot C_{IN}$

$$C_{OUT} = A' \cdot B \cdot C_{IN} + A \cdot B' \cdot C_{IN} + A \cdot B \cdot C_{IN}' + A \cdot B \cdot C_{IN}$$

$$S = A' \cdot B' \cdot C_{IN} + A' \cdot B \cdot C_{IN}' + A \cdot B' \cdot C_{IN}' + A \cdot B \cdot C_{IN}$$

1-bit Binary Adder

- **Inputs:** A, B, Carry-in
- **Outputs:** Sum, Carry-out



A	B	C_{IN}	C_{OUT}	S
0	0	0	0	0
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	1

$$S = A' \cdot B' \cdot C_{IN} + A' \cdot B \cdot C_{IN}' + A \cdot B' \cdot C_{IN}' + A \cdot B \cdot C_{IN}$$

$$C_{OUT} = A' \cdot B \cdot C_{IN} + A \cdot B' \cdot C_{IN} + A \cdot B \cdot C_{IN}' + A \cdot B \cdot C_{IN}$$

Apply Theorems to Simplify Expressions

The theorems of Boolean algebra can simplify expressions

– e.g., full adder's carry-out function

$$
\begin{aligned}
\text{Cout} &= A'\,B\,\text{Cin} + A\,B'\,\text{Cin} + A\,B\,\text{Cin}' + A\,B\,\text{Cin} \\
&= A'\,B\,\text{Cin} + A\,B'\,\text{Cin} + A\,B\,\text{Cin}' + A\,B\,\text{Cin} + A\,B\,\text{Cin} \\
&= A'\,B\,\text{Cin} + A\,B\,\text{Cin} + A\,B'\,\text{Cin} + A\,B\,\text{Cin}' + A\,B\,\text{Cin} \\
&= (A' + A)\,B\,\text{Cin} + A\,B'\,\text{Cin} + A\,B\,\text{Cin}' + A\,B\,\text{Cin} \\
&= (1)\,B\,\text{Cin} + A\,B'\,\text{Cin} + A\,B\,\text{Cin}' + A\,B\,\text{Cin} \\
&= B\,\text{Cin} + A\,B'\,\text{Cin} + A\,B\,\text{Cin}' + A\,B\,\text{Cin} + A\,B\,\text{Cin} \\
&= B\,\text{Cin} + A\,B'\,\text{Cin} + A\,B\,\text{Cin} + A\,B\,\text{Cin}' + A\,B\,\text{Cin} \\
&= B\,\text{Cin} + A\,(B' + B)\,\text{Cin} + A\,B\,\text{Cin}' + A\,B\,\text{Cin} \\
&= B\,\text{Cin} + A\,(1)\,\text{Cin} + A\,B\,\text{Cin}' + A\,B\,\text{Cin} \\
&= B\,\text{Cin} + A\,\text{Cin} + A\,B\,(\text{Cin}' + \text{Cin}) \\
&= B\,\text{Cin} + A\,\text{Cin} + A\,B\,(1) \\
&= B\,\text{Cin} + A\,\text{Cin} + A\,B
\end{aligned}
$$

Apply Theorems to Simplify Expressions

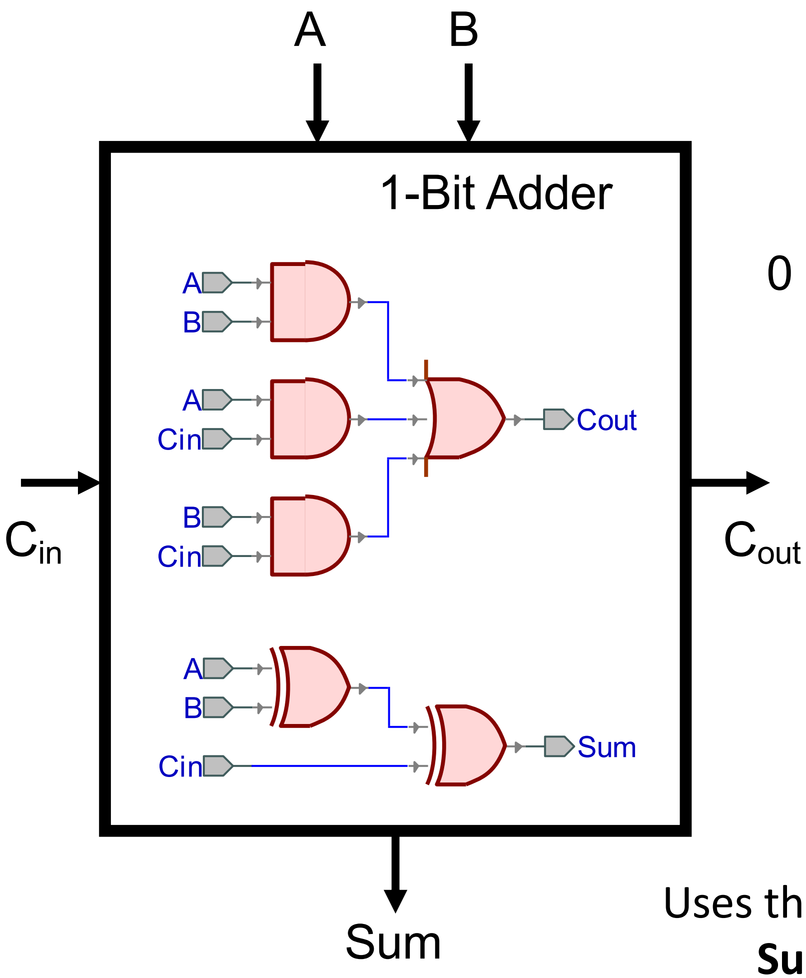
The theorems of Boolean algebra can simplify expressions

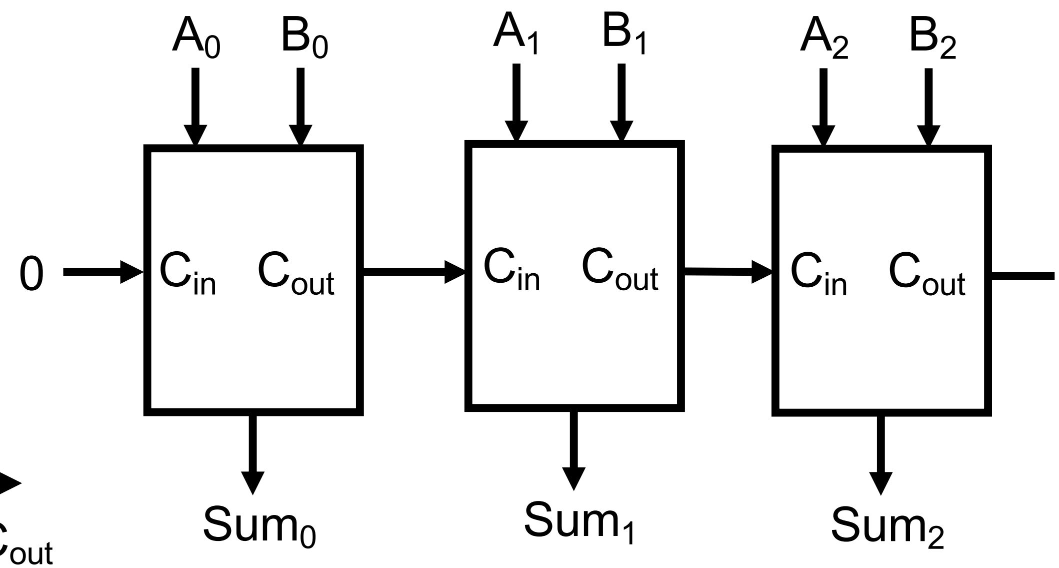
– **e.g., full adder's carry-out function**

$$
\begin{aligned}
Cout &= A'\,B\,Cin + A\,B'\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= A'\,B\,Cin + A\,B'\,Cin + A\,B\,Cin' + \boxed{A\,B\,Cin + A\,B\,Cin} \\
&= A'\,B\,Cin + A\,B\,Cin + A\,B'\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= (A' + A)\,B\,Cin + A\,B'\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= (1)\,B\,Cin + A\,B'\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= B\,Cin + A\,B'\,Cin + A\,B\,Cin' + \boxed{A\,B\,Cin + A\,B\,Cin} \\
&= B\,Cin + A\,B'\,Cin + A\,B\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= B\,Cin + A\,(B' + B)\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= B\,Cin + A\,(1)\,Cin + A\,B\,Cin' + A\,B\,Cin \\
&= B\,Cin + A\,Cin + A\,B\,(Cin' + Cin) \\
&= B\,Cin + A\,Cin + A\,B\,(1) \\
&= B\,Cin + A\,Cin + A\,B
\end{aligned}
$$

adding extra terms creates new factoring opportunities

A 2-bit Ripple-Carry Adder





Uses the fact that

$$\mathbf{Sum} = A' \cdot B' \cdot C_{IN} + A' \cdot B \cdot C_{IN}' + A \cdot B' \cdot C_{IN}' + A \cdot B \cdot C_{IN}$$

is equivalent to $\mathbf{Sum} = (A \oplus B) \oplus C_{IN}$

Mapping Truth Tables to Logic Gates

Given a truth table:

1. Write the output in a table
2. Write the Boolean expression
3. Minimize the Boolean expression
4. Draw as gates
5. Map to available gates

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

(2)

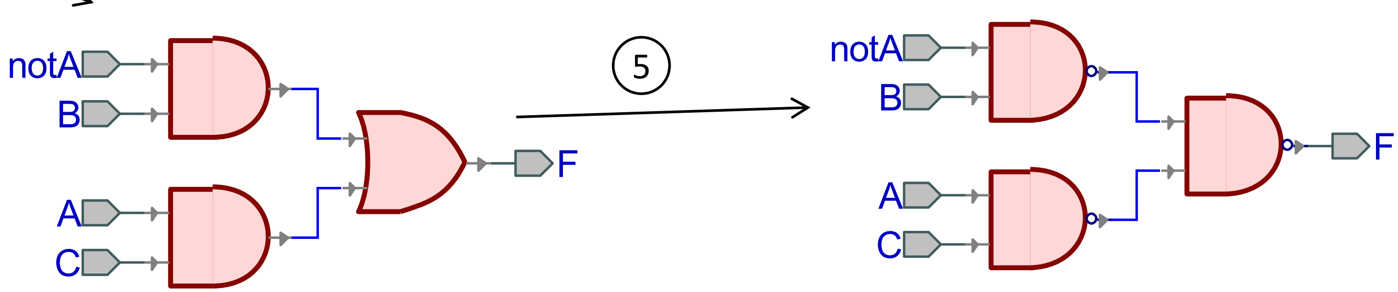
$$F = A'BC'+A'BC+AB'C+ABC$$

(3)

$$= A'B(C'+C)+AC(B'+B)$$

$$= A'B+AC$$

(4)



Canonical Forms

- Truth table is the **unique signature** of a 0/1 function

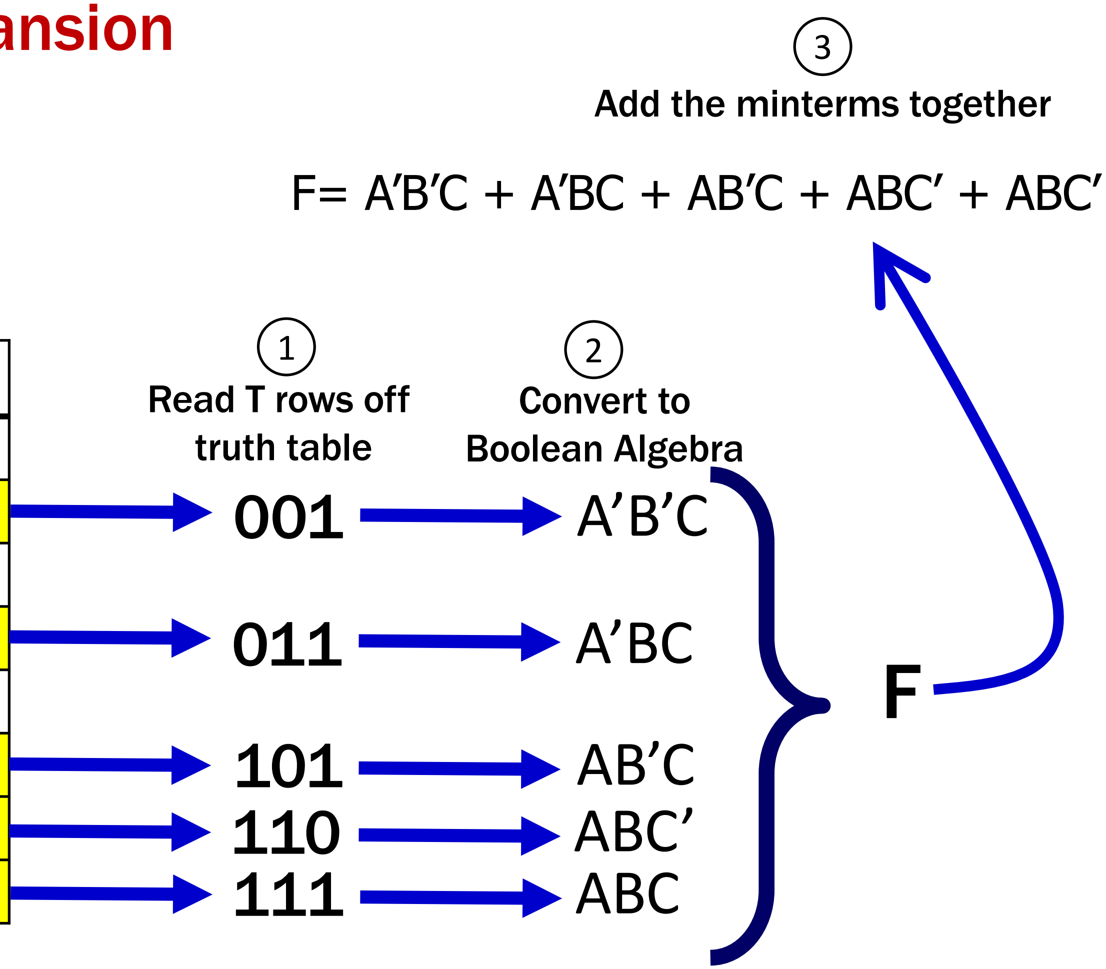
- The same truth table can have many gate realizations
  - We've seen this already
  - Depends on how good we are at Boolean simplification

- Canonical forms
  - Standard forms for a Boolean expression
  - We all produce the same expression

Sum-of-Products Canonical Form

- AKA **Disjunctive Normal Form (DNF)**
- AKA **Minterm Expansion**

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



Sum-of-Products Canonical Form

Product term (or minterm)

- **ANDed product of literals – input combination for which output is true**
- **each variable appears exactly once, true or inverted (but not both)**

A	B	C	minterms
0	0	0	A′B′C′
0	0	1	A′B′C
0	1	0	A′BC′
0	1	1	A′BC
1	0	0	AB′C′
1	0	1	AB′C
1	1	0	ABC′
1	1	1	ABC

F in canonical form:

F(A, B, C) = A′B′C + A′BC + AB′C + ABC′ + ABC

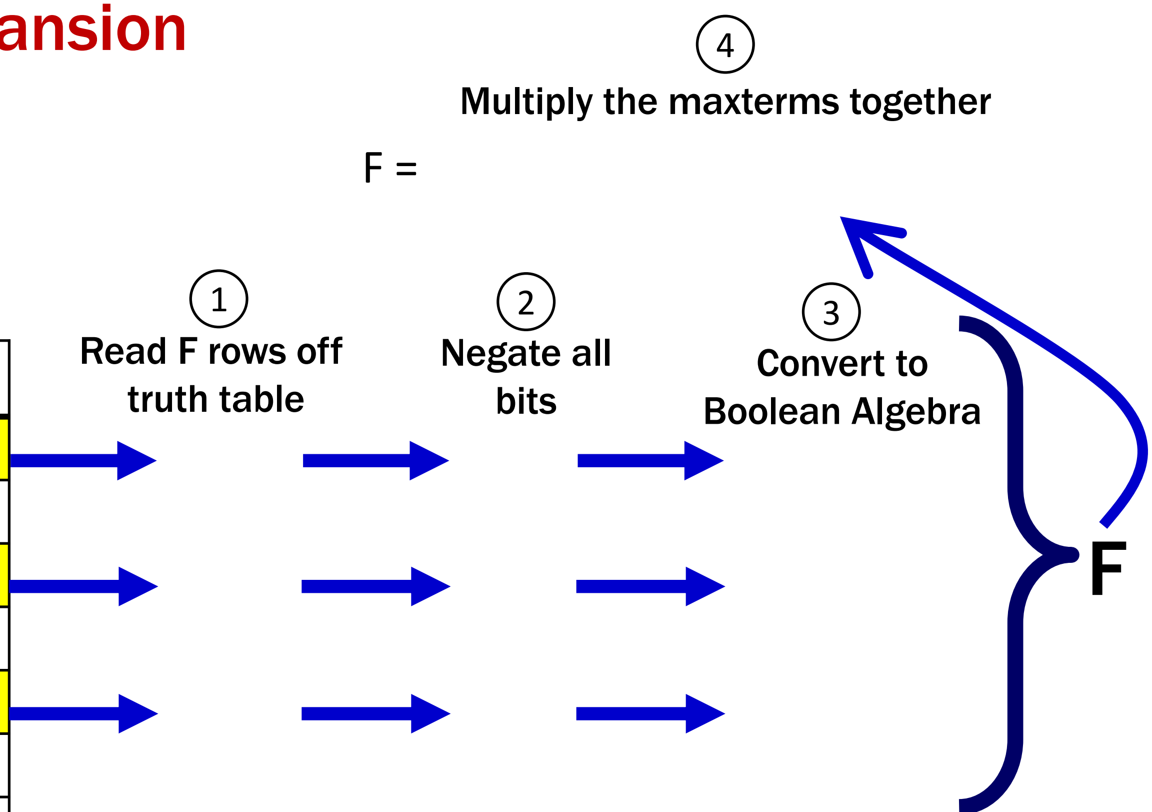
canonical form ≠ minimal form

F(A, B, C) = A′B′C + A′BC + AB′C + ABC + ABC′
= (A′B′ + A′B + AB′ + AB)C + ABC′
= ((A′ + A)(B′ + B))C + ABC′
= C + ABC′
= ABC′ + C
= AB + C

Product-of-Sums Canonical Form

- AKA **Conjunctive Normal Form (CNF)**
- AKA **Maxterm Expansion**

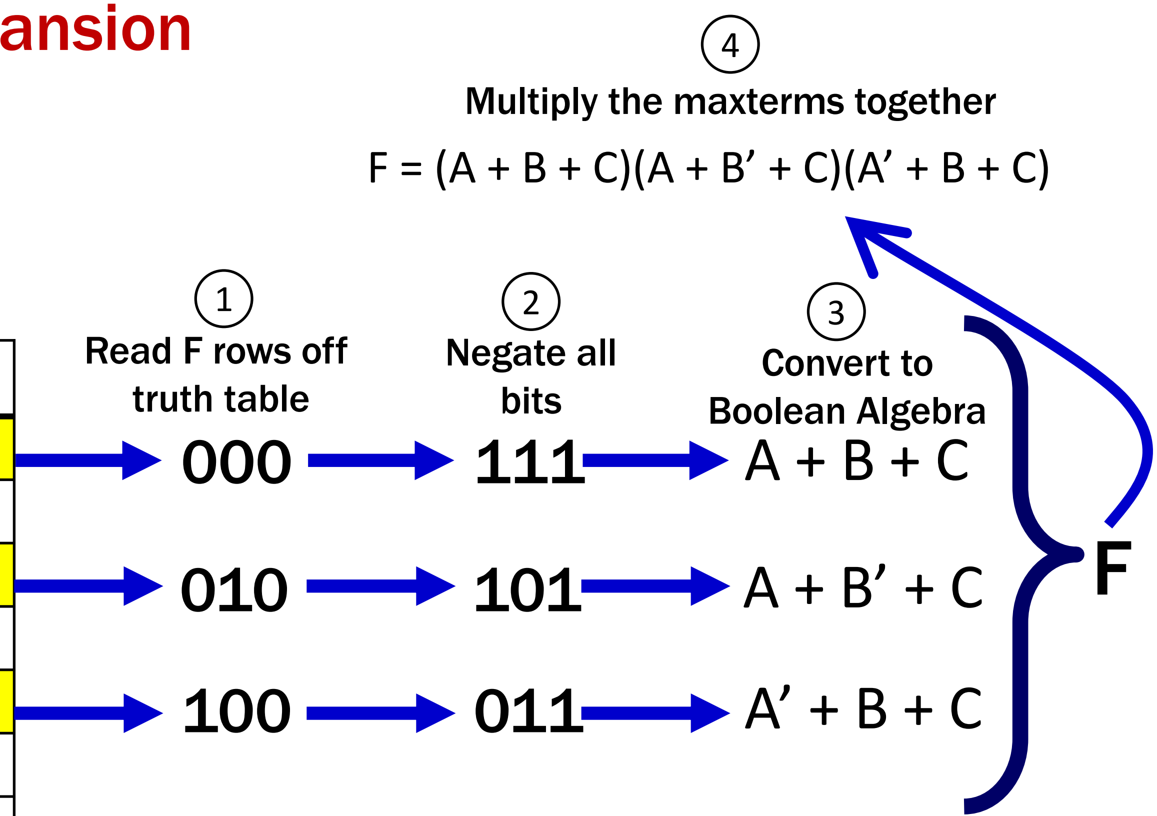
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Product-of-Sums Canonical Form

- AKA **Conjunctive Normal Form (CNF)**
- AKA **Maxterm Expansion**

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



Product-of-Sums: Why does this procedure work?

Useful Facts:

- We know (F')' = F
- We know how to get a **minterm** expansion for F'

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

$$F' = A'B'C' + A'BC' + AB'C'$$

Product-of-Sums: Why does this procedure work?

Useful Facts:

- We know (F')' = F
- We know how to get a **minterm** expansion for F'

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

$$F' = A'B'C' + A'BC' + AB'C'$$

Taking the complement of both sides…

$$(F')' = (A'B'C' + A'BC' + AB'C')'$$

And using DeMorgan/Comp….

$$F = (A'B'C')' \; (A'BC')' \; (AB'C')'$$

$$F = (A + B + C)(A + B' + C)(A' + B + C)$$

Product-of-Sums Canonical Form

Sum term (or maxterm)

- ORed sum of literals – input combination for which output is false
- each variable appears exactly once, true or inverted (but not both)

A	B	C	maxterms
0	0	0	A+B+C
0	0	1	A+B+C′
0	1	0	A+B′+C
0	1	1	A+B′+C′
1	0	0	A′+B+C
1	0	1	A′+B+C′
1	1	0	A′+B′+C
1	1	1	A′+B′+C′

F in canonical form:

$$F(A, B, C) = (A + B + C)\,(A + B' + C)\,(A' + B + C)$$

canonical form ≠ minimal form

$$\begin{aligned} F(A, B, C) &= (A + B + C)\,(A + B' + C)\,(A' + B + C) \\ &= (A + B + C)\,(A + B' + C) \\ &\quad\ (A + B + C)\,(A' + B + C) \\ &= (A + C)\,(B + C) \end{aligned}$$

Predicate Logic

Predicate Logic

- **Propositional Logic**

  "If you take the high road and I take the low road then I'll arrive in Scotland before you."

- **Predicate Logic**

  "All positive integers x, y, and z satisfy $x^3 + y^3 \neq z^3$."

Predicate Logic

- Propositional Logic
  - Allows us to analyze complex propositions in terms of their simpler constituent parts (a.k.a. atomic propositions) joined by connectives

- Predicate Logic
  - Lets us analyze them at a deeper level by expressing how those propositions depend on the objects they are talking about

Predicate Logic

Adds two key notions to propositional logic

- Predicates

- Quantifiers

Predicates

Predicate

– **A function that returns a truth value, e.g.,**

Cat(x) ::= “x is a cat”

Prime(x) ::= “x is prime”

HasTaken(x, y) ::= “student x has taken course y”

LessThan(x, y) ::= “x < y”

Sum(x, y, z) ::= “x + y = z”

GreaterThan5(x) ::= “x > 5”

HasNChars(s, n) ::= “string s has length n”

Predicates can have varying numbers of arguments and input types.

Domain of Discourse

For ease of use, we define one "type"/"domain" that we work over. This non-empty set of objects is called the "**domain of discourse**".

For each of the following, what might the domain be?

(1) "x is a cat", "x barks", "x ruined my couch"

"mammals" or "sentient beings" or "cats and dogs" or …

(2) "x is prime", "x = 0", "x < 0", "x is a power of two"

"numbers" or "integers" or "integers greater than 5" or …

(3) "student x has taken course y" "x is a pre-req for z"

"students and courses" or "university entities" or …