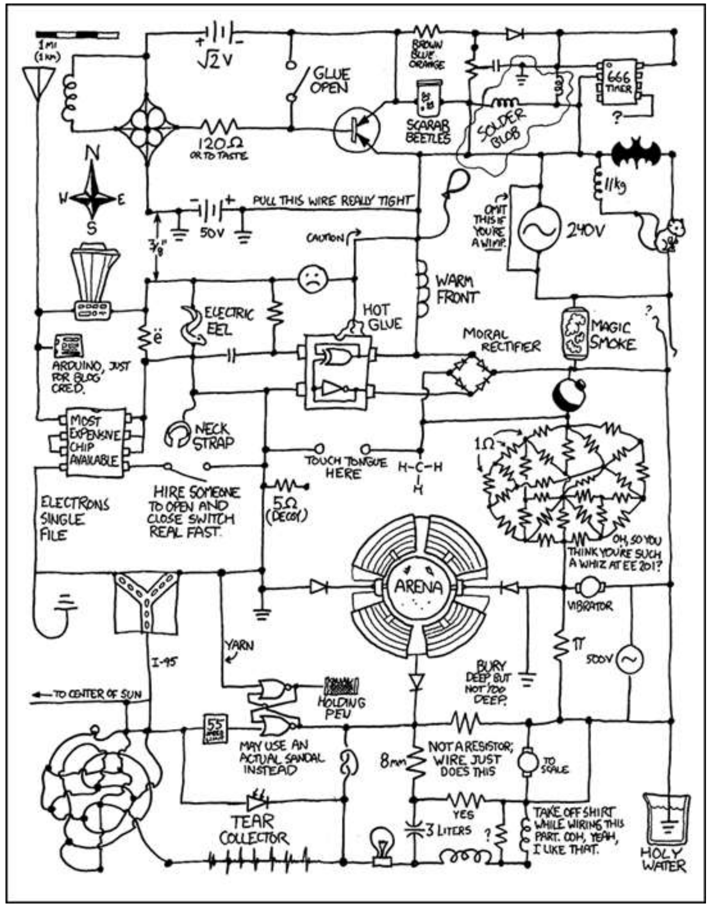
CSE 311: Foundations of Computing

Lecture 5: DNF, CNF and Predicate Logic

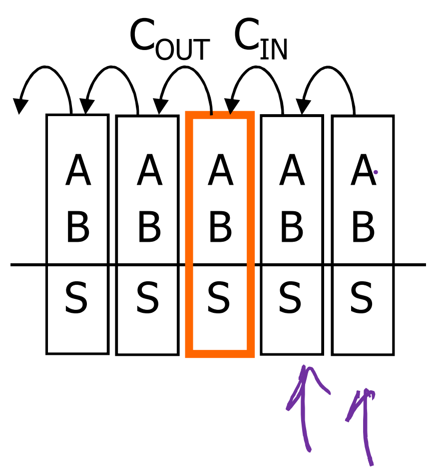


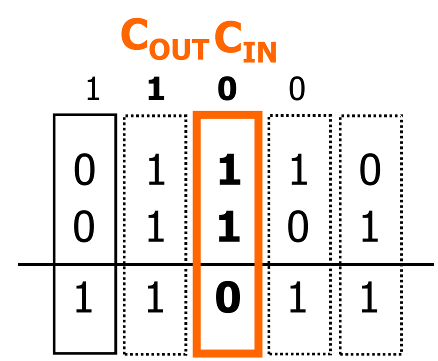
1-bit Binary Adder

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: These are chained together, with a carry-in

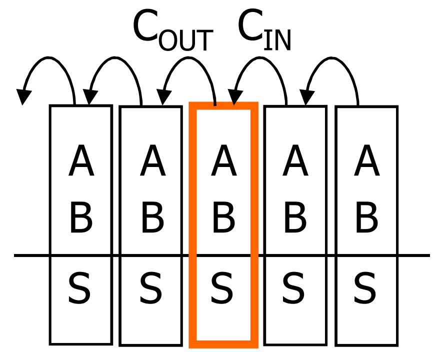
(C_{IN})
A
+ B
S
(C_{OUT})



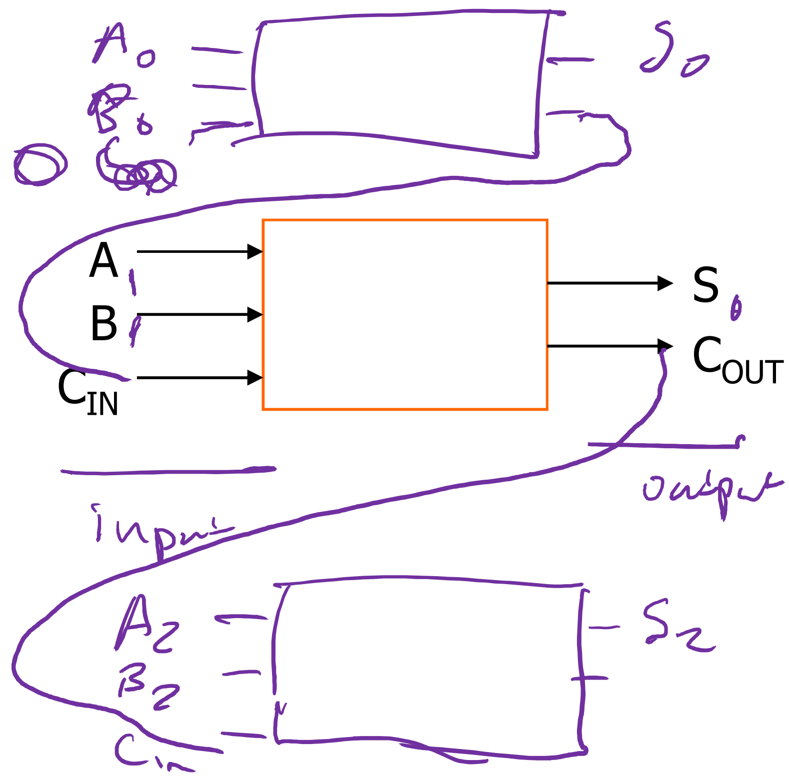


1-bit Binary Adder

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

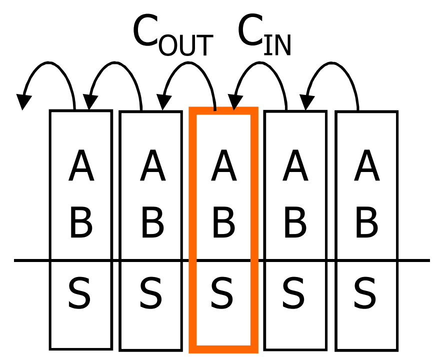


A	B	C_{IN}	C_{OUT}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



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

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

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

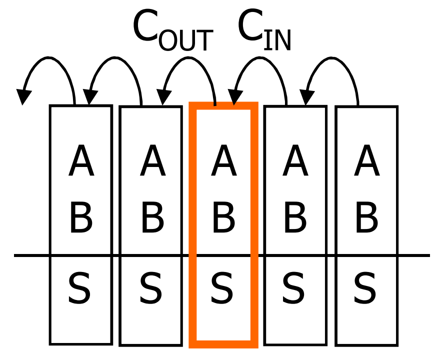
$A \cdot B' \cdot C_{IN}'$

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

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

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}

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

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

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

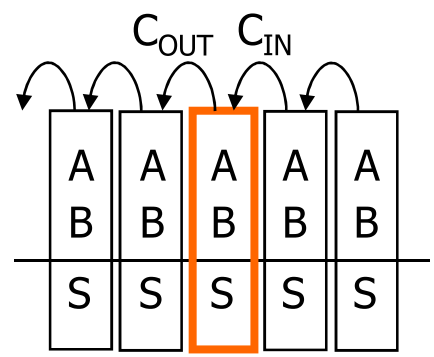
$A \cdot B \cdot C_{IN}$

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

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

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

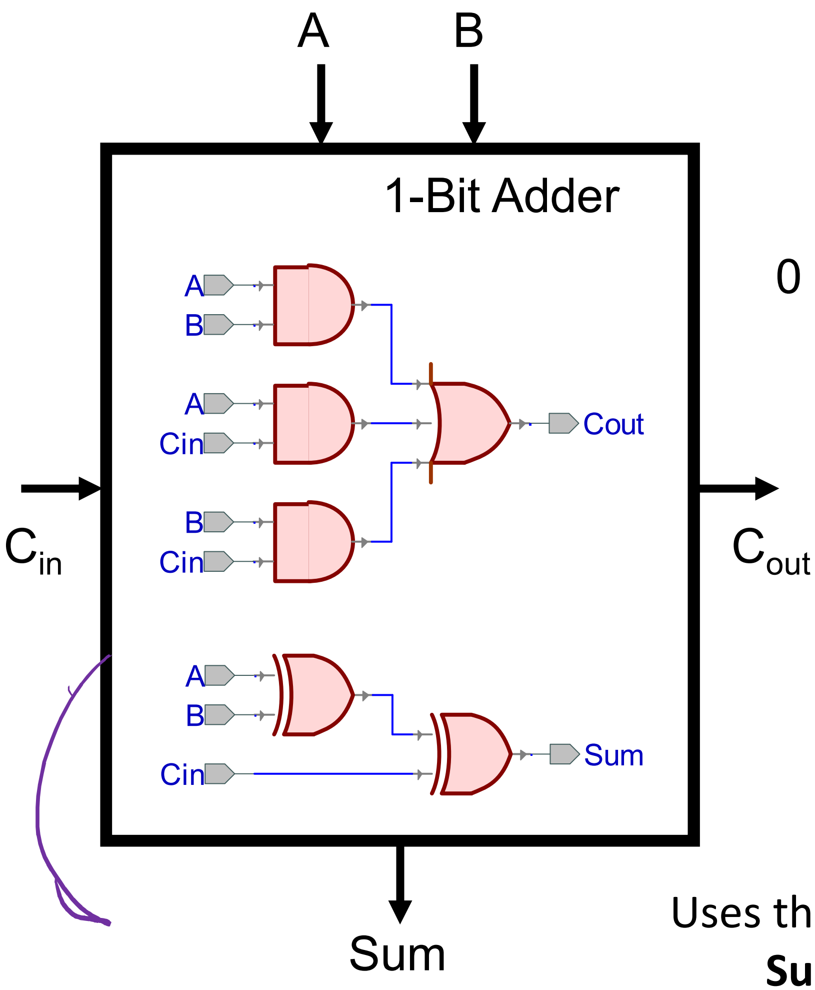
The theorems of Boolean algebra can simplify expressions

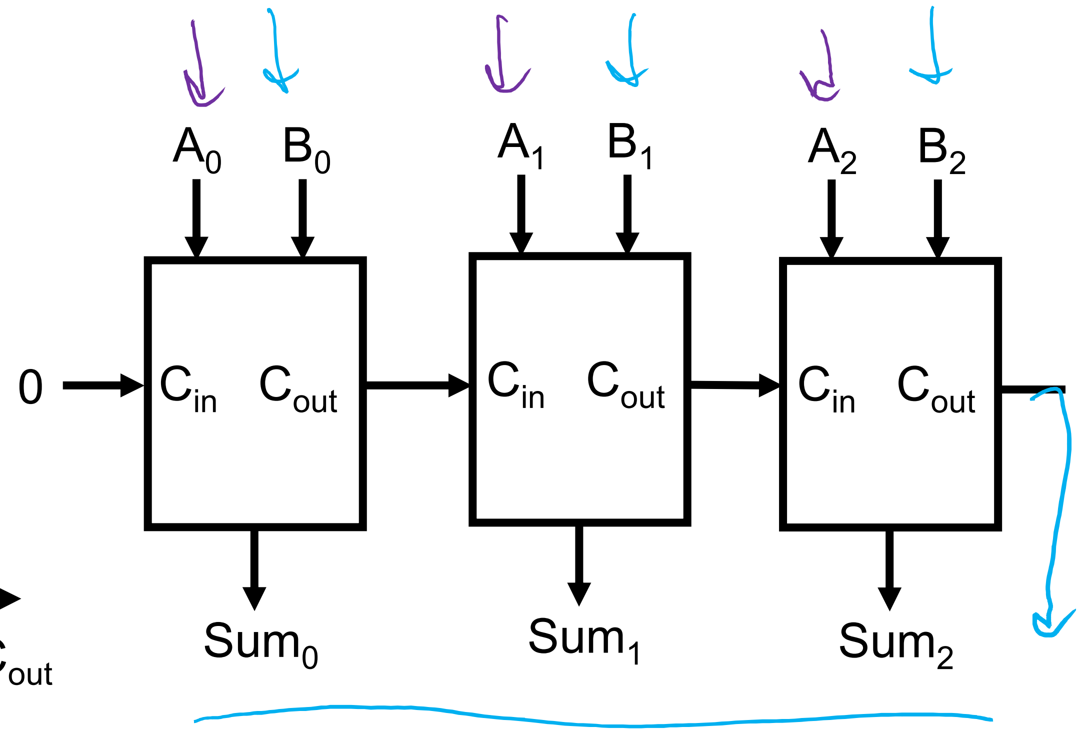
– **e.g., full adder's carry-out function**

Cout = A′ B Cin + A B′ Cin + A B Cin′ + A B Cin
= A′ B Cin + A B′ Cin + A B Cin′ + 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′ + 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

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

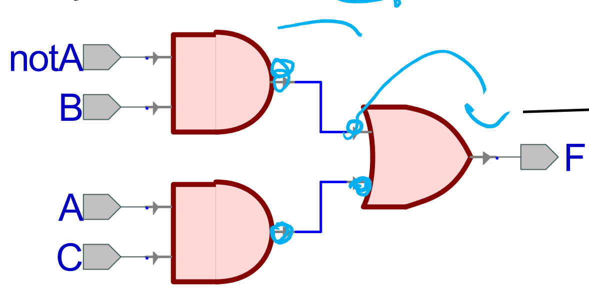
Mapping Truth Tables to Logic Gates

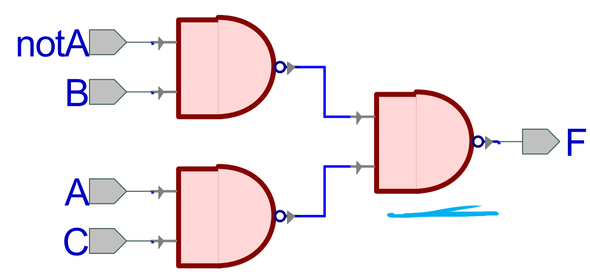
Given a truth table:

1. **Write the Boolean expression**
2. **Minimize the Boolean expression**
3. **Draw as gates**
4. **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

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





$\neg p \vee \neg q \equiv \neg(p \wedge q)$

Canonical Forms

- Truth table is the unique signature of a Boolean 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 come up with the same expression

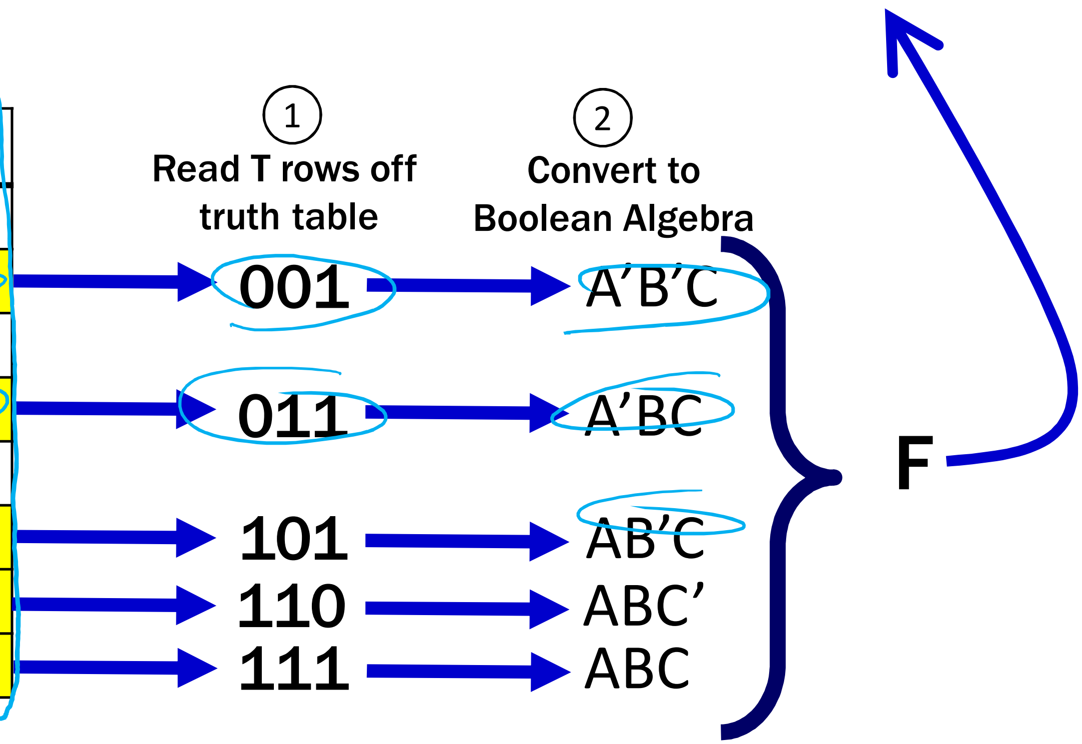
Sum-of-Products Canonical Form

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

③ **Add the minterms together**

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

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:

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

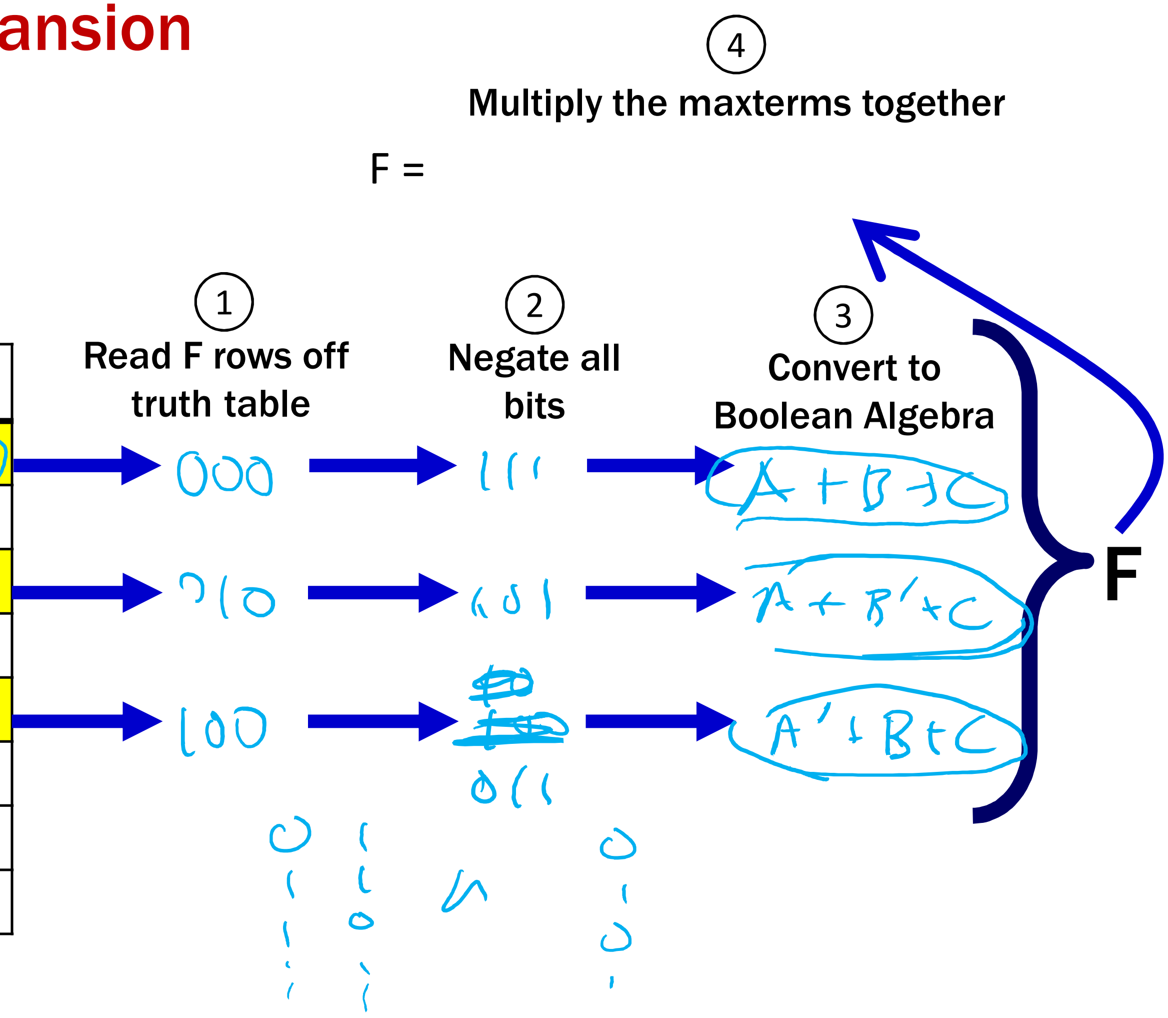
canonical form ≠ minimal form

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

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

① Read F rows off truth table	② Negate all bits	③ Convert to Boolean Algebra
000	111	A + B + C
010	101	A + B' + C
100	011	A' + B + C

F

④ Multiply the maxterms together

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

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

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

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

- **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 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"

pets, mammals

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

numbers, integers

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

students + courses

Domain of Discourse

For ease of use, we define one "type"/"domain" that we work over. This 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 …

Quantifiers

We use *quantifiers* to talk about collections of objects.

$\forall x\ P(x)$

P(x) is true **for every** x in the domain

read as "**for all x, P of x**"

$\exists x\ P(x)$

There is an x in the domain for which P(x) is true

read as "**there exists x, P of x**"

Quantifiers

We use *quantifiers* to talk about collections of objects.

Universal Quantifier ("for all"): $\forall x\ P(x)$

$P(x)$ is true for every x in the domain

read as "for all x, P of x"

Examples:

Are these true?

- $\forall$x Odd(x)
- $\forall$x LessThan5(x)

Quantifiers

We use *quantifiers* to talk about collections of objects.

Universal Quantifier ("for all"): $\forall x\ P(x)$

$P(x)$ is true for every x in the domain

read as "for all x, P of x"

Examples:

Are these true? It depends on the domain. For example:

	{1, 3, -1, -27}	Integers	Odd Integers
• $\forall x$ Odd(x)	True	False	True
• $\forall x$ LessThan4(x)	True	False	False

Quantifiers

We use *quantifiers* to talk about collections of objects.

Existential Quantifier ("exists"): $\exists x\ P(x)$

There is an x in the domain for which P(x) is true

read as "there exists x, P of x"

Examples:

Are these true?

- $\exists x$ Odd(x)
- $\exists x$ LessThan5(x)

Quantifiers

We use *quantifiers* to talk about collections of objects.

Existential Quantifier ("exists"): $\exists x\ P(x)$

There is an x in the domain for which $P(x)$ is true

read as "there exists x, P of x"

Examples:

Are these true? It depends on the domain. For example:

	{1, 3, -1, -27}	Integers	Positive Multiples of 5
• $\exists x$ Odd(x)	True	True	True
• $\exists x$ LessThan4(x)	True	True	False

Statements with Quantifiers

Just like with propositional logic, we need to define variables (this time **predicates**) before we do anything else. We must also now define a **domain of discourse** before doing anything else.

Domain of Discourse
Positive Integers

Predicate Definitions	
Even(x) ::= “x is even”	Greater(x, y) ::= “x > y”
Odd(x) ::= “x is odd”	Equal(x, y) ::= “x = y”
Prime(x) ::= “x is prime”	Sum(x, y, z) ::= “x + y = z”

Statements with Quantifiers

Domain of Discourse
Positive Integers

Predicate Definitions	
Even(x) ::= "x is even"	Greater(x, y) ::= "x > y"
Odd(x) ::= "x is odd"	Equal(x, y) ::= "x = y"
Prime(x) ::= "x is prime"	Sum(x, y, z) ::= "x + y = z"

Determine the truth values of each of these statements:

$\exists x$ Even(x) T

$\forall x$ Odd(x) F

$\forall x$ (Even(x) $\vee$ Odd(x)) T

$\exists x$ (Even(x) $\wedge$ Odd(x)) F

$\forall x$ Greater(x+1, x) T

$\exists x$ (Even(x) $\wedge$ Prime(x)) T

Statements with Quantifiers

Domain of Discourse
Positive Integers

Predicate Definitions	
Even(x) ::= "x is even"	Greater(x, y) ::= "x > y"
Odd(x) ::= "x is odd"	Equal(x, y) ::= "x = y"
Prime(x) ::= "x is prime"	Sum(x, y, z) ::= "x + y = z"

Determine the truth values of each of these statements:

$\exists x$ Even(x)	**T**	**e.g. 2, 4, 6, ...**
$\forall x$ Odd(x)	**F**	**e.g. 2, 4, 6, ...**
$\forall x$ (Even(x) $\vee$ Odd(x))	**T**	**every integer is either even or odd**
$\exists x$ (Even(x) $\wedge$ Odd(x))	**F**	**no integer is both even and odd**
$\forall x$ Greater(x+1, x)	**T**	**adding 1 makes a bigger number**
$\exists x$ (Even(x) $\wedge$ Prime(x))	**T**	**Even(2) is true and Prime(2) is true**

Statements with Quantifiers

Domain of Discourse
Positive Integers

Predicate Definitions	
Even(x) ::= "x is even"	Greater(x, y) ::= "x > y"
Odd(x) ::= "x is odd"	Equal(x, y) ::= "x = y"
Prime(x) ::= "x is prime"	Sum(x, y, z) ::= "x + y = z"

Translate the following statements to English

$\forall x\, \exists y\, \text{Greater}(y, x)$

$\forall x\, \exists y\, \text{Greater}(x, y)$

$\forall x\, \exists y\, (\text{Greater}(y, x) \wedge \text{Prime}(y))$

$\forall x\, (\text{Prime}(x) \rightarrow (\text{Equal}(x, 2) \vee \text{Odd}(x)))$

$\exists x\, \exists y\, (\text{Sum}(x, 2, y) \wedge \text{Prime}(x) \wedge \text{Prime}(y))$

Statements with Quantifiers (Literal Translations)

Domain of Discourse
Positive Integers

Predicate Definitions	
Even(x) ::= "x is even"	Greater(x, y) ::= "x > y"
Odd(x) ::= "x is odd"	Equal(x, y) ::= "x = y"
Prime(x) ::= "x is prime"	Sum(x, y, z) ::= "x + y = z"

Translate the following statements to English

$\forall x\ \exists y$ Greater(y, x)

For every positive integer x, there is a positive integer y, such that y > x.

$\forall x\ \exists y$ Greater(x, y)

For every positive integer x, there is a positive integer y, such that x > y.

$\forall x\ \exists y$ (Greater(y, x) $\wedge$ Prime(y))

For every positive integer x, there is a pos. int. y such that y > x and y is prime.

$\forall x$ (Prime(x) $\rightarrow$ (Equal(x, 2) $\vee$ Odd(x)))

For each positive integer x, if x is prime, then x = 2 or x is odd.

$\exists x\ \exists y$ (Sum(x, 2, y) $\wedge$ Prime(x) $\wedge$ Prime(y))

There exist positive integers x and y such that x + 2 = y and x and y are prime.

Statements with Quantifiers (Natural Translations)

Domain of Discourse
Positive Integers

Predicate Definitions	
Even(x) ::= "x is even"	Greater(x, y) ::= "x > y"
Odd(x) ::= "x is odd"	Equal(x, y) ::= "x = y"
Prime(x) ::= "x is prime"	Sum(x, y, z) ::= "x + y = z"

Translate the following statements to English

$\forall x\ \exists y\ \text{Greater}(y, x)$

There is no greatest positive integer.

$\forall x\ \exists y\ \text{Greater}(x, y)$

There is no least positive integer.

$\forall x\ \exists y\ (\text{Greater}(y, x) \wedge \text{Prime}(y))$

For every positive integer there is a larger number that is prime.

$\forall x\ (\text{Prime}(x) \rightarrow (\text{Equal}(x, 2) \vee \text{Odd}(x)))$

Every prime number is either 2 or odd.

$\exists x\ \exists y\ (\text{Sum}(x, 2, y) \wedge \text{Prime}(x) \wedge \text{Prime}(y))$

There exist prime numbers that differ by two."

English to Predicate Logic

Domain of Discourse
Mammals

Predicate Definitions
Cat(x) ::= “x is a cat”
Red(x) ::= “x is red”
LikesTofu(x) ::= “x likes tofu”

“Red cats like tofu”

“Some red cats don’t like tofu”

English to Predicate Logic

Domain of Discourse
Mammals

Predicate Definitions
Cat(x) ::= "x is a cat"
Red(x) ::= "x is red"
LikesTofu(x) ::= "x likes tofu"

"Red cats like tofu"

$$\forall x\ ((\text{Red}(x) \land \text{Cat}(x)) \rightarrow \text{LikesTofu}(x))$$

"Some red cats don't like tofu"

$$\exists y\ ((\text{Red}(y) \land \text{Cat}(y)) \land \neg\text{LikesTofu}(y))$$

English to Predicate Logic

Domain of Discourse
Mammals

Predicate Definitions
Cat(x) ::= "x is a cat"
Red(x) ::= "x is red"
LikesTofu(x) ::= "x likes tofu"

When putting two predicates together like this, we use an "and".

"Red cats like tofu"

When restricting to a smaller domain in a "for all" we use implication.

When there's no leading quantification, it means "for all".

"Some red cats don't like tofu"

When restricting to a smaller domain in an "exists" we use and.

"Some" means "there exists".

Negations of Quantifiers

Predicate Definitions
PurpleFruit(x) ::= "x is a purple fruit"

(*) $\forall x$ PurpleFruit(x) ("All fruits are purple")

What is the negation of (*)?

(a) "there exists a purple fruit"

(b) "there exists a non-purple fruit"

(c) "all fruits are not purple"

Try your intuition! Which one "feels" right?

Key Idea: In every domain, exactly one of a statement and its negation should be true.

Negations of Quantifiers

Predicate Definitions
PurpleFruit(x) ::= "x is a purple fruit"

(*) $\forall x$ PurpleFruit(x) ("All fruits are purple")

What is the negation of (*)?

(a) "there exists a purple fruit"

(b) "there exists a non-purple fruit"

(c) "all fruits are not purple"

Key Idea: In every domain, exactly one of a statement and its negation should be true.

Domain of Discourse
{plum}

Domain of Discourse
{apple}

Domain of Discourse
{plum, apple}

The only choice that ensures exactly one of the statement and its negation is (b).

De Morgan's Laws for Quantifiers

$$\neg \forall x\, P(x) \equiv \exists x\, \neg P(x)$$

$$\neg \exists x\, P(x) \equiv \forall x\, \neg P(x)$$

De Morgan's Laws for Quantifiers

$$\neg \forall x\, P(x) \equiv \exists x \neg P(x)$$
$$\neg \exists x\, P(x) \equiv \forall x \neg P(x)$$

"There is no largest integer"

$$\neg \exists x \forall y \ (x \geq y)$$
$$\equiv \forall x \neg \forall y \ (x \geq y)$$
$$\equiv \forall x \ \exists y \neg (x \geq y)$$
$$\equiv \forall x \ \exists y \ (x < y)$$

"For every integer there is a larger integer"

Scope of Quantifiers

$\exists x\ (P(x) \land Q(x))$ **vs.** $\exists x\, P(x) \land \exists x\, Q(x)$

Scope of Quantifiers

$\exists x\ (P(x) \wedge Q(x))$ **vs.** $\exists x\, P(x) \wedge \exists x\, Q(x)$

This one asserts P and Q of the *same* x.

This one asserts P and Q of potentially different *x*'s.

Scope of Quantifiers

Example: NotLargest(x) $\equiv \exists$ y Greater (y, x)
$\equiv \exists$ z Greater (z, x)

truth value:

doesn't depend on y **or** z **"bound variables"**

does depend on x **"free variable"**

quantifiers only act on free variables of the formula they quantify

$$\forall x\, (\exists y\, (P(x,y) \rightarrow \forall x\, Q(y, x)))$$

Quantifier "Style"

$$\forall x(\exists y\ (P(x,y) \rightarrow \forall\ x\ Q(y, x)))$$

This isn't "wrong", it's just horrible style.
Don't confuse your reader by using the same variable multiple times…there are a lot of letters…