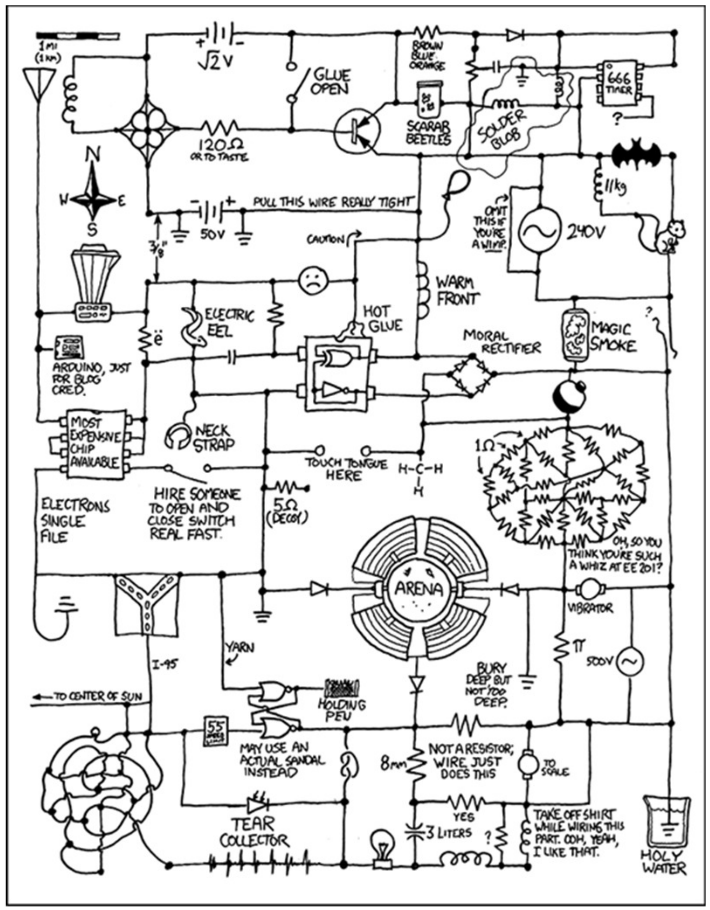
CSE 311: Foundations of Computing

Lecture 5: DNF, CNF and Predicate Logic



xkcd

Administrative

- HW1 due today
  - Submit via Gradescope by 11:00 pm
  - EC1 extra credit submitted separately

- Tomorrow:
  - HW2 out
  - Quiz sections
  - 390Z/ZA sign-up still available

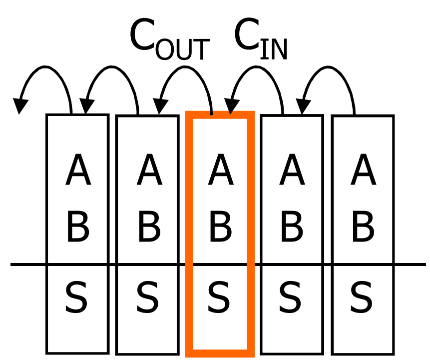
    Loew 113 Thursday 3:30-5:00

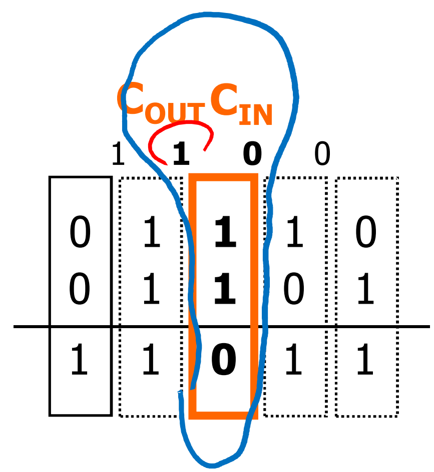
Last Class: 1-bit Binary Adder

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

$$\begin{array}{r} (C_{IN}) \\ A \\ \underline{+\ B} \\ S \\ (C_{OUT}) \end{array}$$





2

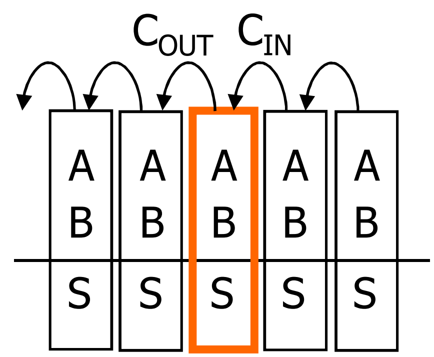
Last Class: Building Boolean Circuits

Design Process:

1. Write down a function table showing desired 0/1 inputs
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 expression to a circuit

Last Class: 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

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

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

Last Class: Apply Theorems to Simplify Expressions

The theorems of Boolean algebra can simplify expressions

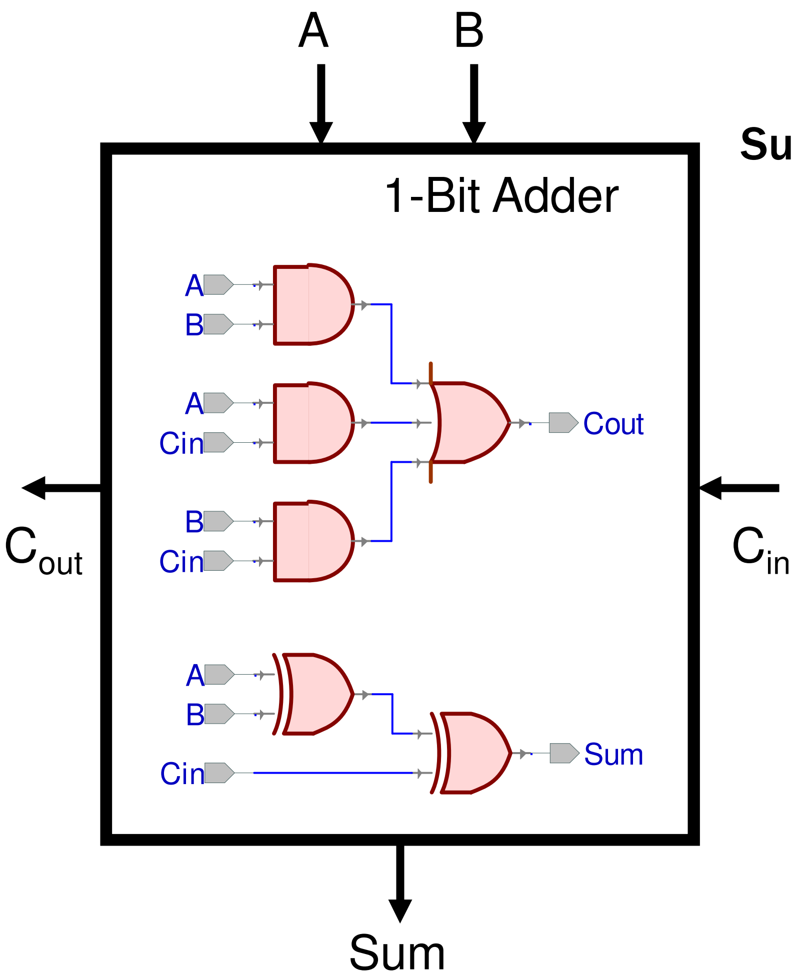
- **e.g., full adder's carry-out function**

Can simplify by combining
with any one of these

Cout = A′ B Cin + A B′ Cin + A B Cin′ + A B Cin
= A′ B Cin + A B′ Cin + A B Cin′ + $\boxed{\text{A B Cin + A B Cin}}$
= 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′ + $\boxed{\text{A B Cin + A B Cin}}$
= 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 copies of
the same term lets us
reuse it for simplification

1-Bit Adder with XOR gates allowed



1 3

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

No Boolean algebra simplifications possible

... but Sum $\equiv (A \oplus B) \oplus C_{IN}$

Mapping Truth Tables to Logic Gates – extra step

Given a truth table:

2. **Write the Boolean expression**
3. **Simplify ("minimize") the Boolean expression**
4. **Draw as gates**
5. ***Map to available gates***

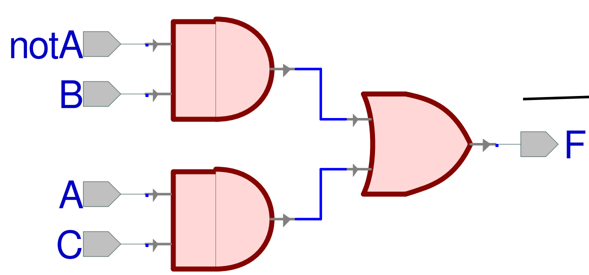
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

1

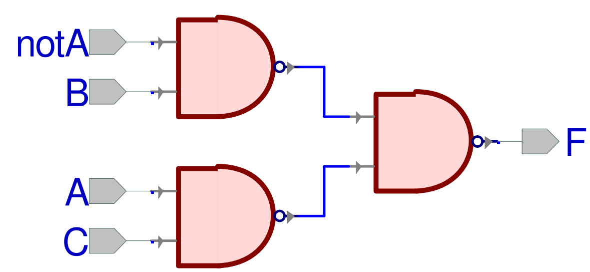
$$
\begin{aligned}
F &= A'BC' + A'BC + AB'C + ABC \\
&= A'B(C'+C) + AC(B'+B) \\
&= A'B + AC
\end{aligned}
$$

2

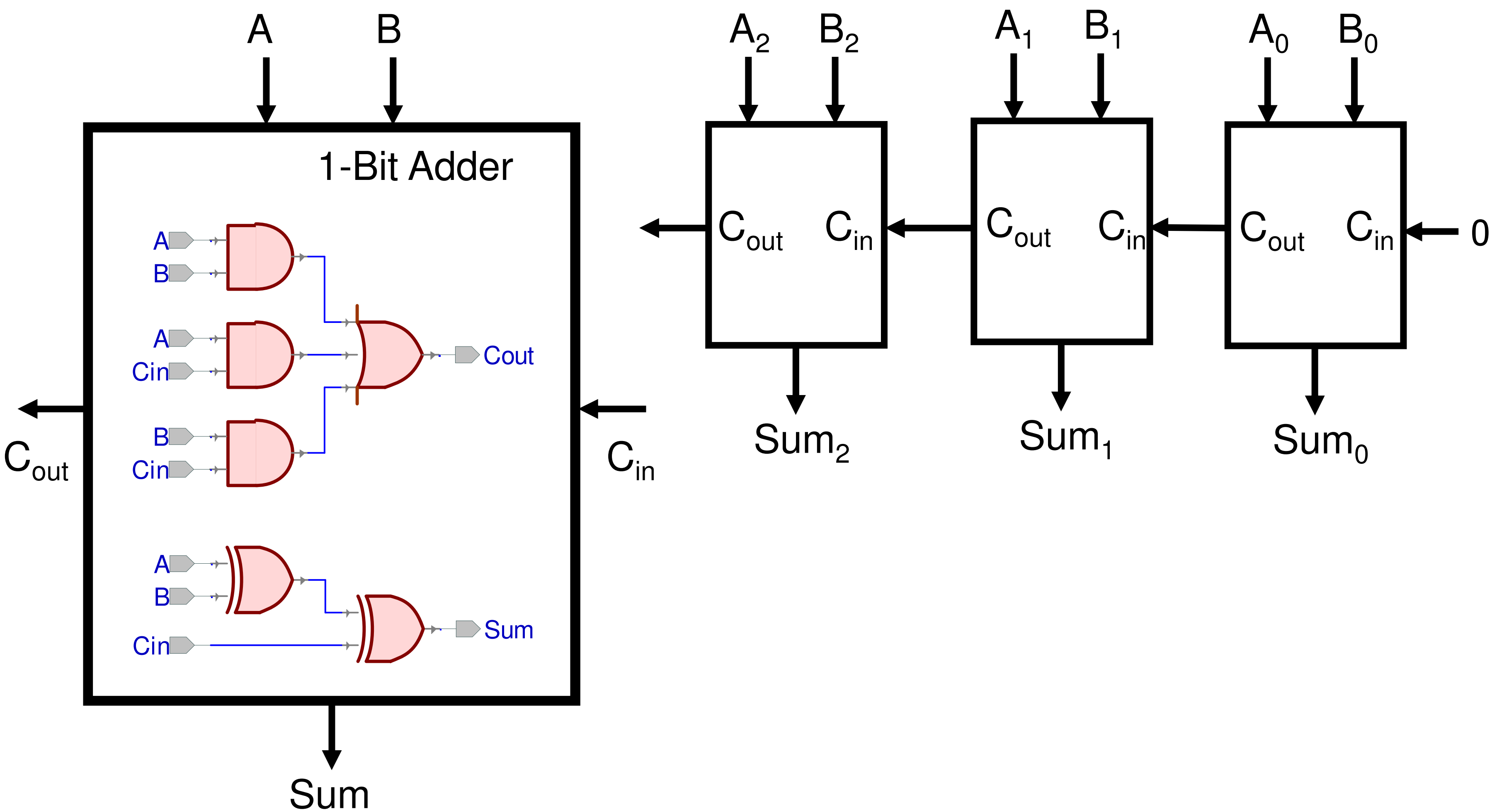
3



4



Multi-bit Ripple-Carry Adder



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

Prop logic

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

Don't simplify!

③ Add the (min)terms 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

① Read T rows off truth table

② Convert to Boolean Algebra

001 → $A'B'C$

011 → $A'BC$

101 → $AB'C$

110 → ABC'

111 → ABC

F

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

Prop logic

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

④ Multiply the maxterms together

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

① Read F rows off truth table

② Negate all bits

③ Convert to Boolean Algebra

F

Product-of-Sums Canonical Form

Prop logic

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

Don't simplify!

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

$000 \rightarrow 111 \rightarrow A + B + C$

$010 \rightarrow 101 \rightarrow A + B' + C$

$100 \rightarrow 011 \rightarrow A' + B + C$

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.…

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

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

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

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”

animals, pets

(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”

Things associated with UW

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 2

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

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

For all x there is a y that is greater than x

T

For every positive integer there is a larger integer

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

F

For every positive integer there's a smaller positive integer

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

T

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

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

T

Every prime is either 2 or is odd

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

5, 7

T

There is a prime number such that there is another prime exactly 2 larger

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

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

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

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

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

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

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

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

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

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

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"

$$\forall x \left(\underbrace{(Cat(x) \land Red(x))} \rightarrow LikesTofu(x) \right)$$

"Some red cats don't like tofu"