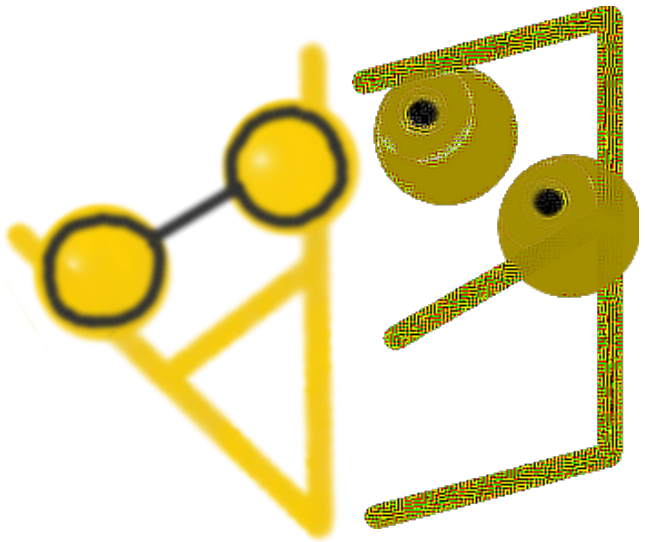
CSE 311



Foundations of Computing I

Fall 2014

Remember, the site is…

http://tinyurl.com/ynlecture

Get started on the first pink handout! (grab TWO pink handouts)

Your task is to find a boolean expression for F.

If you have time, find one for F' as well.

DON'T SIMPLIFY!!!

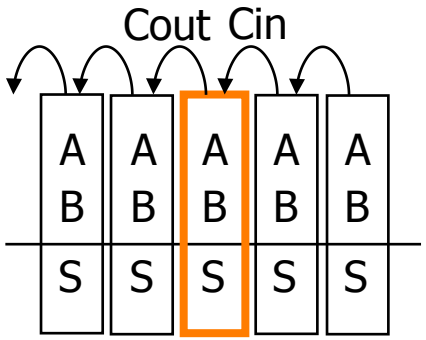
Administrivia and Common Response Stuff

- I'm seeing a lot of "notation mixing" (0 for T, ~ for not, ! for not, etc.) This is actually a problem!
- No questions on handouts lately! Keep on asking! ☹
- How was section yesterday?

- HW back in class on Monday! If you come ~5 minutes early, you can get it back a little early

1-bit Binary Adder

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



A	B	Cin	Cout	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

$$S = A' B' Cin + A' B Cin' + A B' Cin' + A B Cin$$

$$Cout = A' B Cin + A B' Cin + A B Cin' + A B Cin$$

Apply Theorems to Simplify Expressions

The theorems of Boolean algebra can simplify expressions

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

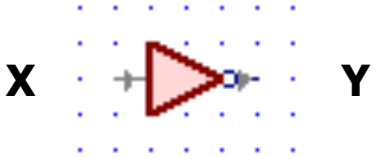
Cout = A′ B Cin + A B′ Cin + A B Cin′ + A B Cin
= A′ B Cin + A B′ Cin + A B Cin′ + A B Cin + A B Cin
= A′ B Cin + A B Cin + A B′ Cin + A B Cin′ + A B Cin
= (A′ + A) B Cin + A B′ Cin + A B Cin′ + A B Cin
= (1) B Cin + A B′ Cin + A B Cin′ + A B Cin
= B Cin + A B′ Cin + A B Cin′ + A B Cin + A B Cin
= B 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

Gates Again!

NOT

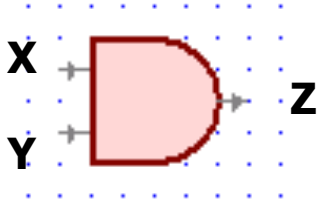
X' $\bar{X}$ $\neg X$



X	Y
0	1
1	0

AND

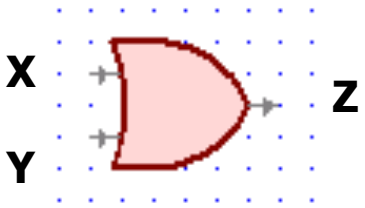
$X \cdot Y$ XY $X \wedge Y$



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

OR

$X + Y$ $X \vee Y$

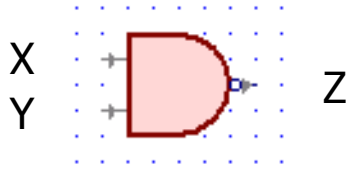


X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

More Gates!

NAND

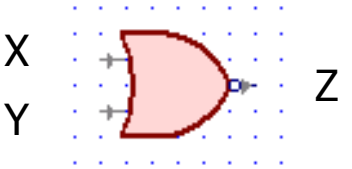
$\neg(X \wedge Y)$ $(XY)'$



X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

NOR

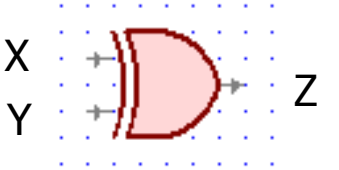
$\neg(X \vee Y)$ $(X + Y)'$



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

XOR

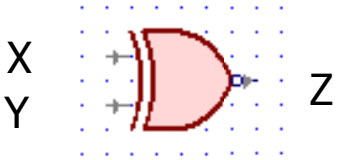
$X \oplus Y$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

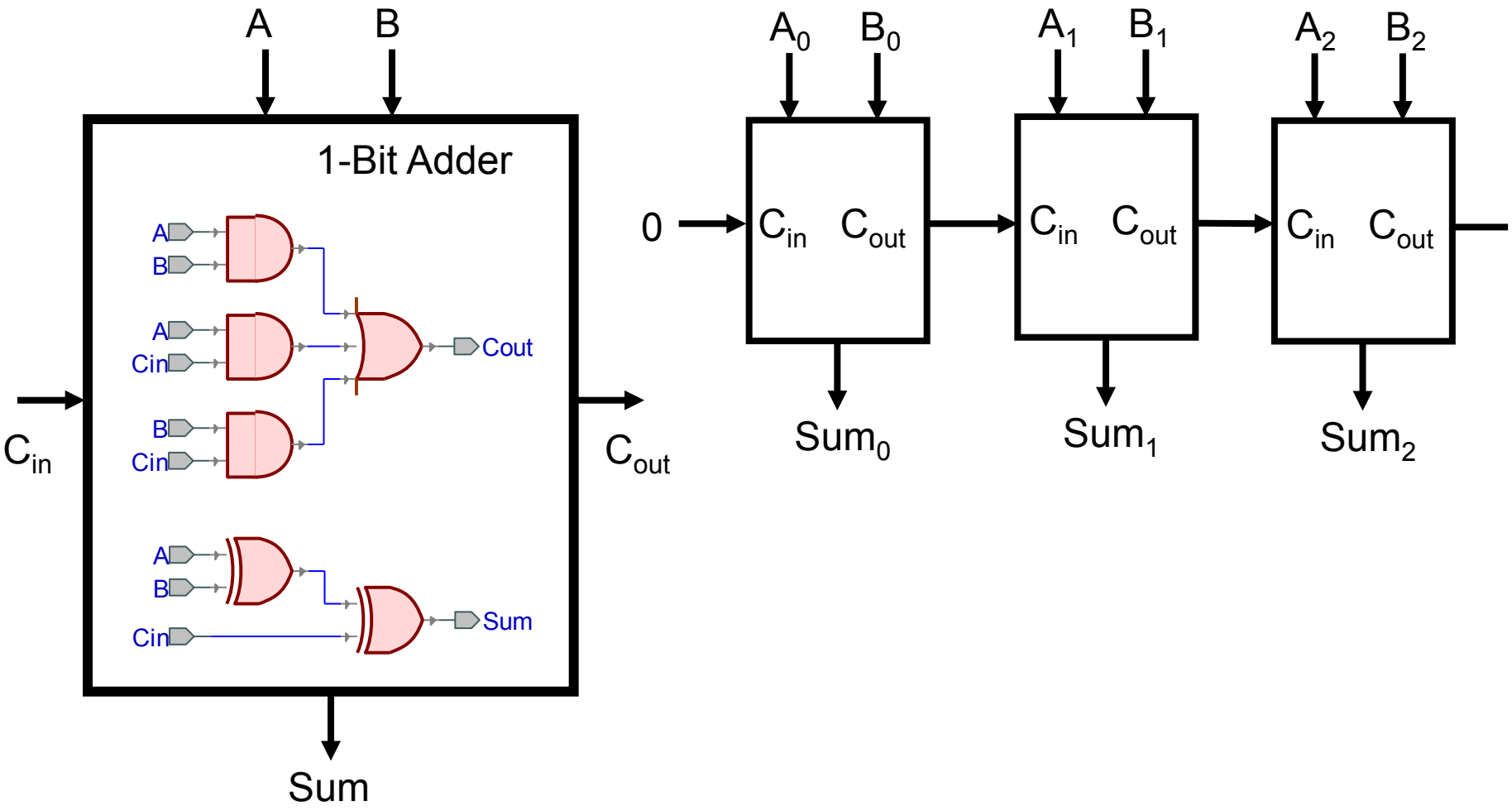
XNOR

$X \leftrightarrow Y$



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

A 2-bit Ripple-Carry Adder



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

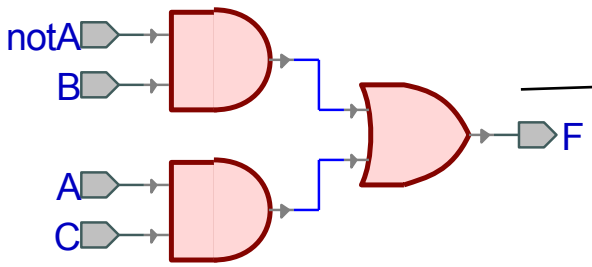
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

(1)

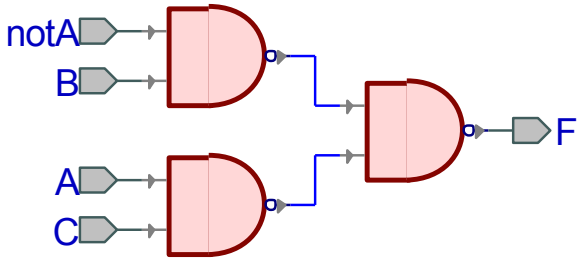
(2)

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

(3)



(4)



Canonical Forms

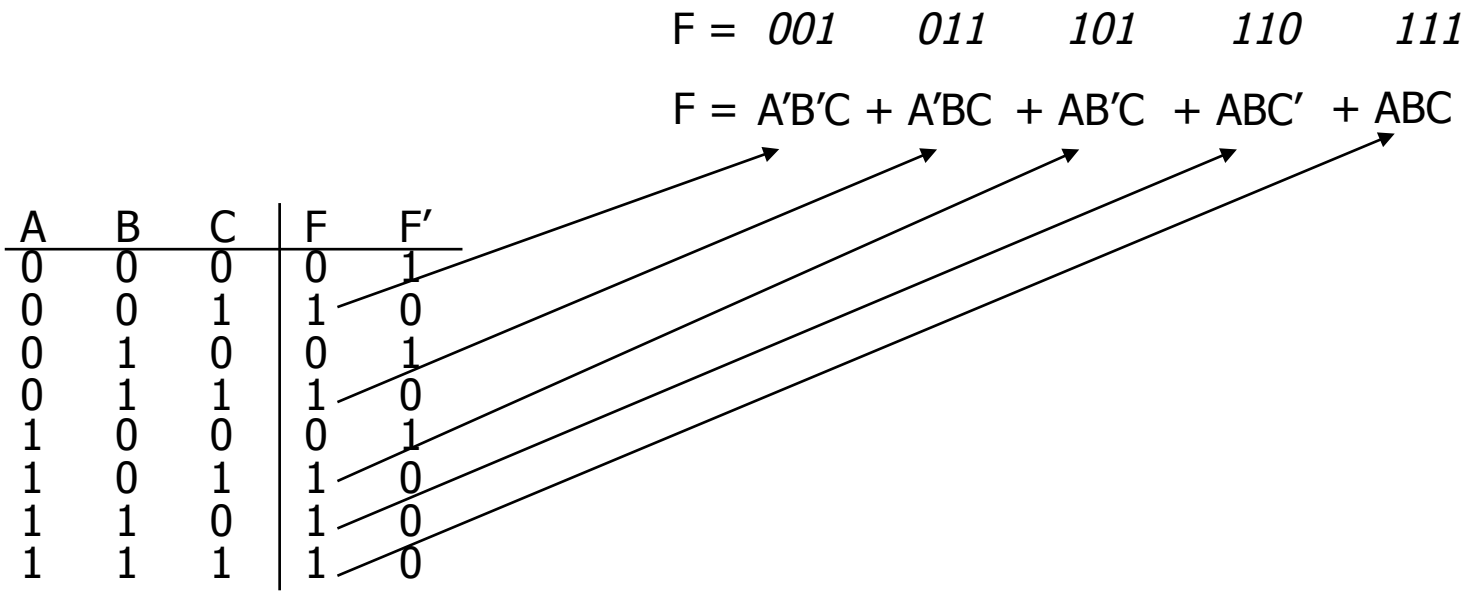
- Truth table is the unique signature of a Boolean function

- 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

- also known as **Disjunctive Normal Form (DNF)**
- also known as **minterm expansion**



A	B	C	F	F′
0	0	0	0	1
0	0	1	1	0
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

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:

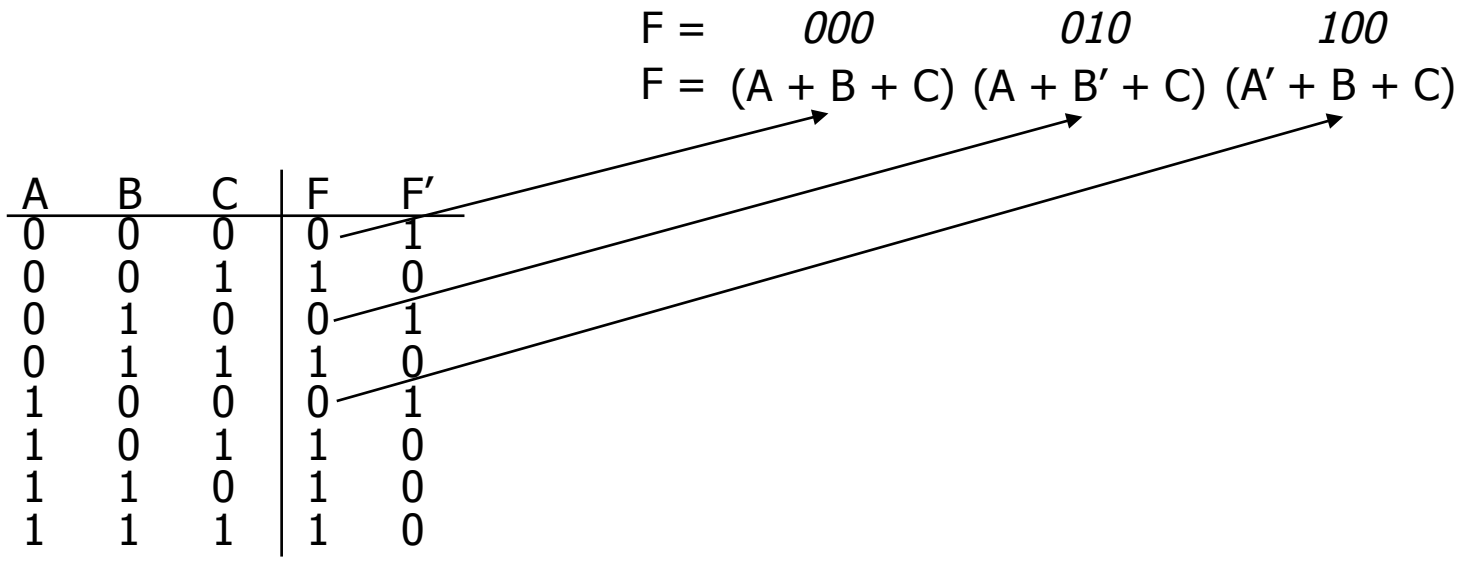
$F(A, B, C) = A'B'C + A'BC + AB'C + ABC' + ABC$

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

- Also known as **Conjunctive Normal Form (CNF)**
- Also known as **maxterm expansion**



s-o-p, p-o-s, and de Morgan's theorem

Complement of function in sum-of-products form:

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

Complement again and apply de Morgan's and get the product-of-sums form:

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

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:

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

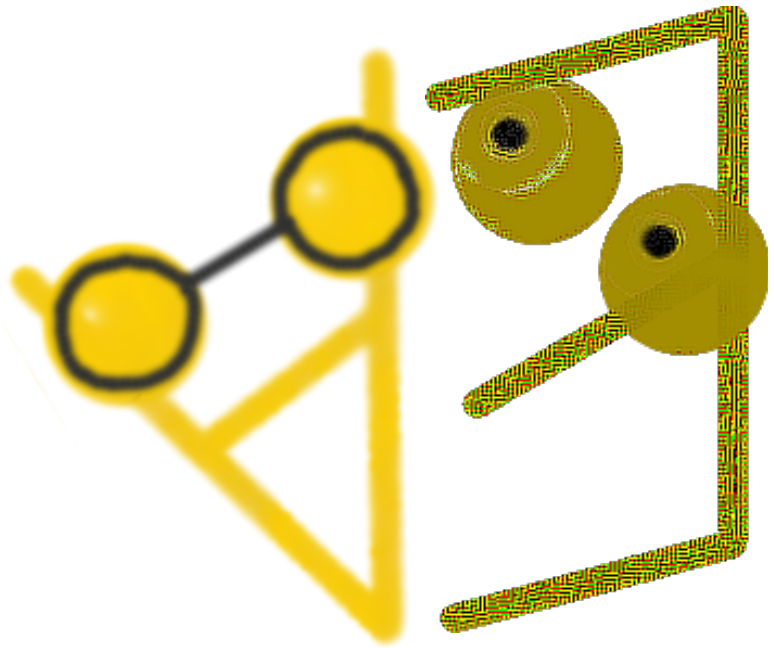
canonical form ≠ minimal form

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

CSE 311: Foundations of Computing

Fall 2014

Lecture 5: Canonical Forms, Predicate Logic



Predicate Logic

- **Propositional Logic**
  - If the tortoise walks at a rate of one node per step, and the hare walks at a rate of two nodes per step, ...

- **Predicate Logic**
  - If the tortoise is on node x, and the hare is on node 2x, then ...

Predicate Logic

- **Propositional Logic**
  - Allows us to analyze complex propositions in terms of their simpler constituent parts joined by connectives

- **Predicate Logic**
  - Lets us analyze them at a deeper level…

Predicate Logic

Predicate or **Propositional Function**

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

"x is a cat"

"x is prime"

"student x has taken course y"

"x > y"

"x + y = z" or Sum(x, y, z)

"5 < x"

Predicates will have **variables** or **constants** as arguments.

Domain of Discourse

We must specify a "domain of discourse", which is the possible things we're talking about.

"x is a cat"

(e.g., mammals)

"x is prime"

(e.g., numbers)

student x has taken course y"

(e.g., students and courses)

Quantifiers

$\forall x\ P(x)$

$P(x)$ 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"

Statements with Quantifiers

- $\exists x$ Even(x)
  - There is an even positive integer.
- $\forall x$ Odd(x)
  - All positive integers are odd.
- $\forall x$ (Even(x) $\vee$ Odd(x))
  - All positive integers are either even or odd.
- $\exists x$ (Even(x) $\wedge$ Odd(x))
  - There is a positive integer that is both even and odd.
- $\forall x$ Greater(x+1, x)
  - For all positive integers, x, x+1 > x.
- $\exists x$ (Even(x) $\wedge$ Prime(x))
  - There is a positive integer which is even and prime.

Domain:
Positive Integers

Even(x)
Odd(x)
Prime(x)
Greater(x,y)
(or "x>y")
Equal(x,y)
(or "x=y")
Sum(x,y,z)

Statements with Quantifiers

- $\forall x \exists y$ Greater (y, x)
  - For every positive integer x, there is a positive integer, y, s.t. y > x.
- $\forall x \exists y$ Greater (x, y)
  - For every positive integer x, there is a positive integer, y, s.t. x > y.
- $\forall x \exists y$ (Greater(y, x) $\wedge$ Prime(y))
  - For every positive integer x, there is a positive integer, y, s.t. x > y and y is prime.
- $\forall x$ (Prime(x) $\rightarrow$ (Equal(x, 2) $\vee$ Odd(x)))
  - For every 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 s. t. x + y = 2 and x and y are both prime.

Domain:
Positive Integers

Even(x)
Odd(x)
Prime(x)
Greater(x,y)
(or "x>y")
Equal(x,y)
(or "x=y")
Sum(x,y,z)

English to Predicate Logic

- "Red cats like tofu"
  - $\forall x ((Red(x) \wedge Cat(x)) \rightarrow LikesTofu(x)$
- "Some red cats don't like tofu"
  - $\exists y ((Red(y) \wedge Cat(y)) \wedge \neg LikesTofu(y))$

Cat(x)
Red(x)
LikesTofu(x)

Domain is mammals!

Negations of Quantifiers

- not every positive integer is prime
- some positive integer is not prime
- prime numbers do not exist
- every positive integer is not prime

Negations of Quantifiers

- ∀x PurpleFruit(x)
  - **"All fruits are purple"**
- What is ¬∀x PurpleFruit(x)?
  - **"Not all fruits are purple"**

Domain:
Fruit

PurpleFruit(x)

- How about ∃x PurpleFruit(x)**?**
  - **"There is a purple fruit"**
  - **If it's the negation, all situations should be covered by a statement and its negation**
  - **Consider the domain** {Orange}**: Neither statement is true!**
  - **No!**

- How about ∃x ¬PurpleFruit(x)**?**
  - **"There is a fruit that isn't purple"**
  - **Yes!**

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"

$$\begin{aligned} \neg \exists x \forall y \; (x \geq y) &\equiv \forall x \neg \forall y \; (x \geq y) \\ &\equiv \forall x \; \exists y \; \neg (x \geq y) \\ &\equiv \forall x \; \exists y \; (y > x) \end{aligned}$$

"For every integer there is a larger integer"

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

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

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.

Nested Quantifiers

- **bound variable names don't matter**

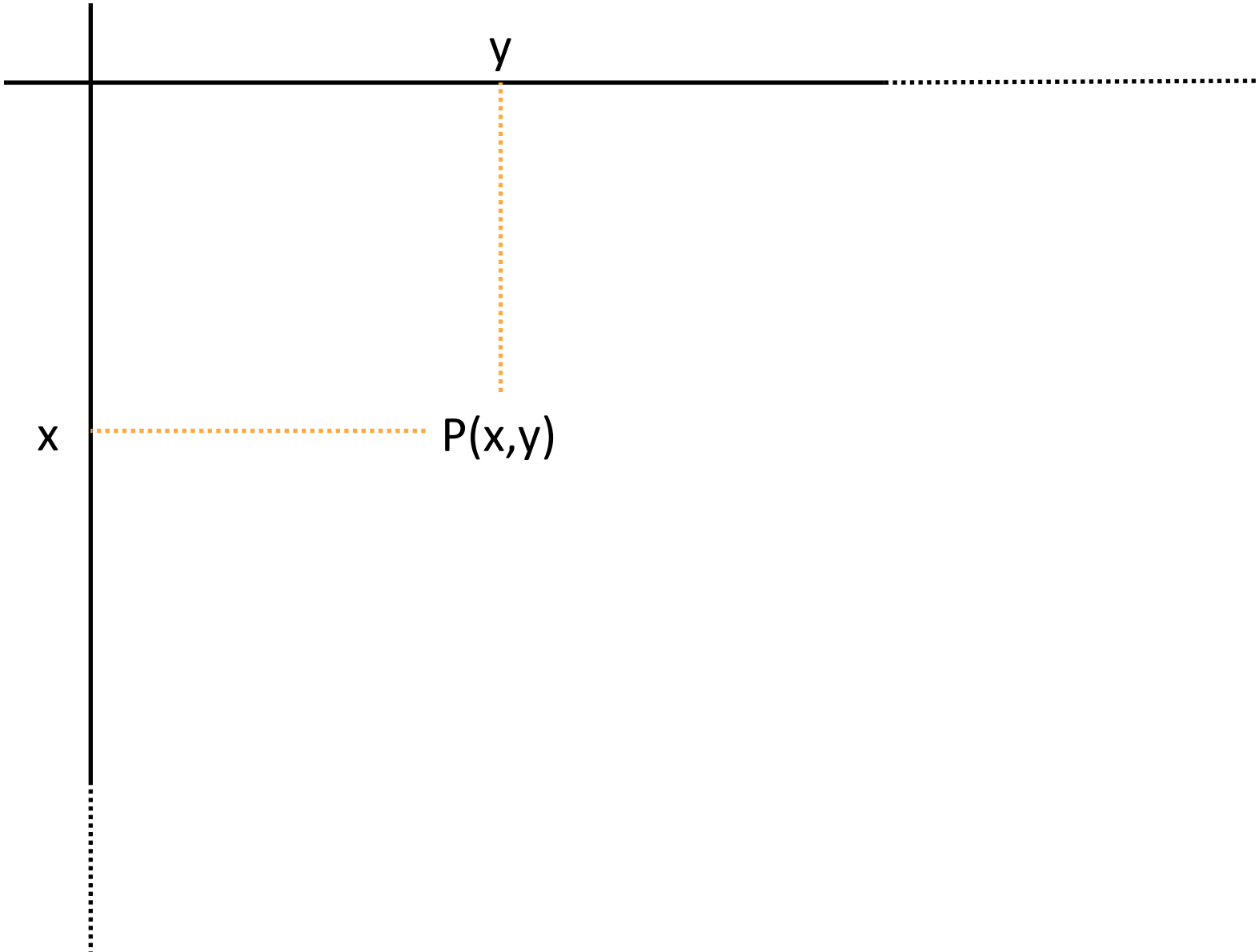
$$\forall x \exists y P(x, y) \equiv \forall a \exists b P(a, b)$$

- **positions of quantifiers can sometimes change**

$$\forall x (Q(x) \wedge \exists y P(x, y)) \equiv \forall x \exists y (Q(x) \wedge P(x, y))$$

- **but: order is important...**

Predicate with Two Variables



Quantification with Two Variables

expression	when **true**	when **false**
$\forall x \forall y P(x, y)$		
$\exists x \exists y P(x, y)$		
$\forall x \exists y P(x, y)$		
$\exists y \forall x P(x, y)$		

Turtles All The Way Down

If the tortoise walks at a rate of one node per step, and the hare walks at a rate of two nodes per step, then the distance between them increases by one node per step.

If the tortoise is on node x, and the hare is on node 2x, then the distance between them increases by one node per step

OnNode(x)

Domain:
Non-negative Integers