COMP S264F Unit 1: Logic

Dr. Keith Lee

School of Science and Technology

The Open University of Hong Kong

Course Aim & Learning Outcomes

- This course aims to lay the foundation of discrete mathematics of students which will be used in studying other computing courses.
- **Learning outcomes:**

1. *Formulate* a range of problems in computing using the notions of set and function;
2. *Explain* and *apply* logical notations and different proof techniques to solve discrete mathematical problems;
3. *Identify* different types of counting problems and *apply* combinatorial methods to solve these problems;
4. *Analyze* mathematical problems involving random processes using probability theory.

- See the Course Guide for the course contents, assessment scheme and textbook.

More about this course

- Cover the foundational structures for the practice of computer science and engineering.

Fundamental tasks in computer science	**How this course would help you?**
Translate imprecise specification into a working system	Precise, reliable and powerful thinking
Get the details right and giving formal proofs for them	Ability to state and prove non-trivial facts
Apply well-known algorithms to your problems	Familiarity with logic, combinatorics, discrete probabilities that are concepts underlying all more advanced courses in computer science.

- Use **Python programs** to help you understand conceptual materials.

Overview

- Propositions, Compound propositions
- Logical operators: $\neg$, $\wedge$, $\vee$, $\otimes$, $\rightarrow$, $\leftrightarrow$
- Truth tables
- Tautology, Contradiction, Logical equivalence
- A logical puzzle
- De Morgan’s laws
- Predicates, Universal quantifier, Existential quantifier

Logic

- It is about a set of rules, which gives precise meaning to mathematical statements and forms the basis of all mathematical reasoning.

- A good understanding of logic allows us to distinguish between valid and invalid mathematical argument.

Logic (cont')

- Proofs in computer science and mathematics require a precisely stated proposition to be proved.

- Natural language is imprecise:

1. You fail the course when your overall continuous assessment score (OCAS) is less than 40 **or** your exam score is less than 40.

➔ You fail the course if both of your OCAS **and** exam scores are less than 40. Correct or not?

2. You are now in the main campus (MC) **or** the Jockey club campus (JCC).

➔ You are now in MC **and** JCC. Correct or not?

Basic terminology: Propositions

- A **proposition** is a statement that is either **true** or **false**, but **not** both.

- Which of the following statements are propositions?
  - Hong Kong is a special administrative region of China.
  - 2 + 2 = 3.
  - What time is it?
  - Read this carefully.
  - X + 1 = 4.

Compound Propositions

- A **compound** proposition is formed from propositions using ***logical operators***.
- Example:
  "711 is a prime number" **and** "2 + 2 = 4"
- Other logical operators:
  negation, and, or, exclusive or, implication, biconditional

Compound Propositions (cont')

- Let ***p*** and ***q*** be two propositions.

- Negation (not) $\neg p$ *(read as "**not** p")*
- Conjunction (and) $p \wedge q$
- Disjunction (or) $p \vee q$
- Exclusive or $p \otimes q$
- Implication $p \rightarrow q$
- Biconditional $p \leftrightarrow q$

- A logical operator can be **defined** precisely using a **truth table**, which displays the relationships between the truth values of a compound proposition and that of its constituting propositions.

Negation, And, Or, Exclusive Or

- **Truth table** for $\neg$(negation), $\wedge$(and), $\vee$(or), $\otimes$(exclusive or):

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \otimes q$
T	T	F	T	T	F
T	F	F	F	T	T
F	T	T	F	T	T
F	F	T	F	F	F

T: true
F: false

- A truth table lists out all the possible values of different logical statements in different scenarios.

Examples

- True or False?
- “Today is Tuesday” $\otimes$ “Yesterday is Monday”
- “Today is Tuesday” $\otimes$ “Today is not Tuesday”
- “Today is in October” $\otimes$ “Christmas is in November”

Implication

- The implication $p \to q$ (or $p \Rightarrow q$) is the proposition that is **false** when *p is true and q is false*, and **true** otherwise.

- Truth table:

p	q	$p \to q$
T	T	T
T	**F**	**F**
F	T	T
F	F	T

- $p \to q$ is often read as "p **implies** q", or "**if** p, **then** q".

Examples

True or False?

- If Vanessa is a woman, then she is the programme leader of Computing.
- If Keith is a woman, then she is the king of China.
- Keith’s age > 10 $\Rightarrow$ 1+1 = 3
- YC’s age < 10 $\Rightarrow$ 1+1 = 2

Double Implication (Biconditional)

- The biconditional $\boldsymbol{p \leftrightarrow q}$ (or $p \Leftrightarrow q$) is the **true** when both p and q have the same truth value, and is **false** otherwise.

- Truth table:

$\boldsymbol{p}$	$\boldsymbol{q}$	$\boldsymbol{p \leftrightarrow q}$
T	**T**	**T**
T	F	F
F	T	F
F	**F**	**T**

- $p \leftrightarrow q$ is often read as "p **if and only if** q", or "p is necessary and sufficient for q", or "p is equivalent to q".

Examples

True or False?

Let A = YC’s age + Keith’s age.

- A is even $\leftrightarrow$ A+1 is odd
- A is prime $\leftrightarrow$ A+1 is odd

Examples

True or False?

Let A = YC's age + Keith's age.

- A is even $\leftrightarrow$ A+1 is odd
  - Case 1: If A is even, then A+1 is odd.
  - Case 2: If A is odd, then A+1 is even.
  - **True**

- A is prime $\leftrightarrow$ A+1 is odd
  - **True** if A = 63 (i.e., "A is prime" is false; "A+1 is odd" is false).
  - **False** if A = 79 (i.e., "A is prime" is true; "A+1 is odd" is false).

Tautology, Contradiction

- Let P be a compound proposition made up of the propositions $p_1, p_2, \ldots, p_n$.
- P is called a **tautology** if P is always **true** for any $p_1, p_2, \ldots, p_n$.
- E.g., $p \vee \neg p$

  $(p \wedge \neg p) \rightarrow q$

- Intuitively, a tautology is a proposition whose **structure** guarantees its truth. The truth values of individual propositions do not matter.

- P is called a **contradiction** if P is always **false** for any $p_1, p_2, \ldots, p_n$.

Logical Equivalence

- Let P and Q be compound propositions made up of the propositions $p_1, p_2, \ldots, p_n$.
- P is said to be logical equivalent to Q, if P and Q always have the same truth value for any $p_1, p_2, \ldots, p_n$.
  Example. P: $p \wedge p$
  Q: p
- Notation. $P \equiv Q$

- Alternate definition: $P \leftrightarrow Q$ is always true for any $p_1, p_2, \ldots, p_n$.

Example:

- Is $p_1 \wedge p_2 \equiv p_1$?

Example

- Is $p_1 \wedge p_2 \equiv p_1$? No.
- If p_1 denotes "10 > 1" and p_2 denotes "10 < 20", then
  $p_1 \wedge p_2 \leftrightarrow p_1$
  $\equiv \mathrm{T} \wedge \mathrm{T} \leftrightarrow \mathrm{T}$
  $\equiv \mathrm{T} \leftrightarrow \mathrm{T}$
  $\equiv \mathrm{T}$

- If p_1 denotes "10 > 1" and p_2 denotes "10 < 2", then
  $p_1 \wedge p_2 \leftrightarrow p_1$
  $\equiv \mathrm{T} \wedge \mathrm{F} \leftrightarrow \mathrm{T}$
  $\equiv \mathrm{F} \leftrightarrow \mathrm{T}$
  $\equiv \mathrm{F}$

Different forms, but same meaning

- Prove that P is (logical) equivalent to Q.
- Prove that $P \equiv Q$.
- Prove that $P \leftrightarrow Q$ is always true for all p_1, p_2, …, p_n.
- Prove that $P \leftrightarrow Q$ is a tautology.

Examples

True or False?

- $\neg\ ((3=4 \lor 4=4) \land (3=2 \lor 2=2))$
- Solution:

  $\neg\ ((3=4 \lor 4=4) \land (3=2 \lor 2=2))$

$\equiv\ \neg\ ((F \lor T) \land (F \lor T))$

$\equiv\ \neg\ (T \land T) \equiv \neg\ T \equiv\ F$

- $4=6 \rightarrow 3=3$
- $(3=2 \lor 4=4) \rightarrow \neg\ (4=4)$
- $3=4 \land \neg\ (2=4 \lor 3=3)$

Examples (cont')

- $4=6 \rightarrow 3=3$

$\equiv F \rightarrow T$

$\equiv T$

- $(3=2 \vee 4=4) \rightarrow \neg (4=4)$

$\equiv (F \vee T) \rightarrow \neg T$

$\equiv T \rightarrow F \equiv F$

- $3=4 \wedge \neg (2=4 \vee 3=3)$

$\equiv F \wedge \neg (F \vee T)$

$\equiv F$

Proof with Truth Table

- Prove that the following propositions are tautology.
  - $(p \rightarrow q) \equiv (\neg q \rightarrow \neg p)$
  - $(\neg\, p \vee q) \equiv (p \rightarrow q)$

- How to prove such a claim? Use a **Truth Table**.

p	q	$\neg p$	$\neg q$	$p \rightarrow q$	$\neg q \rightarrow \neg p$	$\neg\, p \vee q$
T	T	F	F	T	T	T
T	F	F	T	F	F	F
F	T	T	F	T	T	T
F	F	T	T	T	T	T

Set up Python Environment

- This course's Python programs are Jupyter Notebook files (.ipynb files), and will be put on this GitHub repository: https://github.com/cskeith/COMPS264F

- You may run our Python programs on your computer:
  1. Install **Anaconda**: https://www.anaconda.com/products/individual
  2. Anaconda comes with **Jupyter Notebook**, which can be used to open and run .ipynb files.
  3. But I recommend running .ipynb files in **Visual Studio Code**: https://code.visualstudio.com/download
  4. You may find some tutorials on the Internet, e.g., https://code.visualstudio.com/docs/python/jupyter-support

- They can also be run online (but cannot be saved) interactively at: https://mybinder.org/v2/gh/cskeith/COMPS264F/master

Propositions in Python

- Python supports truth values: **True**, **False**
- However, only three logical operators are provided:

Logical operator	Example
and	x **and** y
or	x **or** y
not	**not** x

- Biconditional $\leftrightarrow$: Use the Python comparison operator **==**. E.g., (x **==** y)
- Exclusive or $\otimes$, Implication $\rightarrow$: Define as follows:

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

➢ $p \rightarrow q \equiv (\neg\, p \vee q)$ [By Slide 23]

Generating Truth Table by Python

- To generate the truth tables of exclusive or, implication:

unit01.ipynb

```
def xor(p, q):
    return (p and not q) or (not p and q)

def implies(p, q):
    return not p or q

def table1():
    print( "      p      q  |    xor implies" )
    print( "----------------+---------------" )
    for p in [True, False]:
        for q in [True, False]:
            print("%7r%7r  |%7r%7r" %
                  (p, q, xor(p, q), implies(p, q) ) )

table1()
```

Generating Truth Table by Python (cont')

- Another way is to **use 0, 1** to replace False, True:

unit01.ipynb

```
def xor(p, q):
    return (p and not q) or (not p and q)

def implies(p, q):
    return not p or q

def table2():
    print( "  p  q  | xor implies" )
    print( "--------+------------" )
    for p in [1, 0]:
        for q in [1, 0]:
            print("%3d%3d  |%3d%3d" %
                  (p, q, xor(p, q), implies(p, q) ) )

table2()
```

Puzzle

- In the middle of the journey to afterlife, you need to select whether to go East or West at a branch.
- One is the path to hell and the other is to heaven, but you cannot tell which is which.
- A knowledgeable man called Tom knows the way. Yet you are informed that Tom either always tells the truth or always lies.
- You are allowed to ask Tom to determine the way to heaven. What to ask?

> If you can ask two questions, the problem is trivial.
> Let *P* be the proposition "East is the way to heaven".
> - Question 1: $4 > 5$?
> - Question 2: Is *P* true or false?

- You are allowed to ask only one question!

Solving the Puzzle

- Let P = “East is the way to heaven”, and let Q = “**Tom always lies**”.
- Question: Is “P * Q” true or false? (* is an unknown operator.)

➢Case 1: Tom always tells the truth (Q = false).

$(P * Q) \equiv (P * \text{false})$

➢Case 2: Tom always lies (Q = true).

$\neg (P * Q) \equiv \neg (P * \text{true})$

- Aim: In either case, we want Tom’s answer to reflect P’s truth value.

Is it possible that

$P * \text{false} \equiv P$ and $P * \text{true} \equiv \neg P$?

Solving the Puzzle

- Let P = "East is the way to heaven", and let Q = "**Tom always lies**".
- Question: Is "P * Q" true or false? (* is an unknown operator.)
- Case 1: Tom always tells the truth (Q = false).

  $(P * Q) \equiv (P * \text{false})$
- Case 2: Tom always lies (Q = true).

  $\neg (P * Q) \equiv \neg (P * \text{true})$

- Aim: In either case, we want Tom's answer to reflect P's truth value.

Is it possible that

$$P * \text{false} \equiv P \quad \text{and} \quad P * \text{true} \equiv \neg P \;?$$

Yes! Use exclusive OR. $P \otimes \text{false} \equiv P$; $P \otimes \text{true} \equiv \neg P$

- Question for Tom: Is "(East is the way to heaven) $\otimes$ (You always lie)" true or false?

De Morgan's laws

- **Theorem** $\neg(p \vee q) \equiv (\neg p \wedge \neg q)$
  $\neg(p \wedge q) \equiv (\neg p \vee \neg q)$

- *Again, we can use a truth table to prove the laws.*

p	q	$\neg p$	$\neg q$	$\neg(p \vee q)$	$(\neg p \wedge \neg q)$	$\neg(p \wedge q)$	$(\neg p \vee \neg q)$
T	T	F	F	F	F	F	F
T	F	F	T	F	F	T	T
F	T	T	F	F	F	T	T
F	F	T	T	T	T	T	T

More equivalence

Distributive laws:

- $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$
- $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

Associative laws:

- $p \vee (q \vee r) \equiv (p \vee q) \vee r$
- $p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r$

Commutative laws:

- $p \vee q \equiv q \vee p$
- $p \wedge q \equiv q \wedge p$

Trivial equivalence:

- $p \vee \mathrm{T} \equiv \mathrm{T}$
- $p \vee \mathrm{F} \equiv p$
- $p \wedge \mathrm{T} \equiv p$
- $p \wedge \mathrm{F} \equiv \mathrm{F}$
- $p \vee p \equiv p$
- $p \wedge p \equiv p$
- $\neg(\neg p) \equiv p$
- $p \vee \neg p \equiv \mathrm{T}$
- $p \wedge \neg p \equiv \mathrm{F}$

Example

Prove that $(p \wedge q) \rightarrow (p \vee q) \equiv$ true.

Proof.

$(p \wedge q) \rightarrow (p \vee q)$

$\equiv \neg (p \wedge q) \vee (p \vee q)$ [as $a \rightarrow b \equiv (\neg a) \vee b$]

$\equiv (\neg p \vee \neg q) \vee (p \vee q)$ [De Morgan's law]

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

$\equiv$ true $\vee$ true

$\equiv$ true

Ambiguity

Let ***p*** denote the proposition "if X > 3, then X < 5".

What is the truth value of ***p***?

- In a daily conversation, if somebody mentions ***p*** to you, you probably say "NO" or false.
- Why? Because we assume that ***p*** means

  "No matter what X is, if X > 3, then X < 5".

- Denote the above statement as ***q***.
- Of course, ***q*** is false since when X = 10,

  $X > 3 \rightarrow X < 5$ is false.

Predicates

- “X > 3” is neither true nor false unless the value of X is specified.
- Let P(x) denote the statement “x > 3”.
- Then we can say that P(2) is false, P(5) is true, etc.

- P(x) is called a propositional function; once the value of x is fixed, P(x) has a value – either true or false.
- P is also called the **predicate** (dictionary meaning: the part of a sentence that is not a subject), i.e., greater than 3.

Propositional functions with 2 or more variables

- Let $Q(x, y)$ denote the statement "$x = y + 3$".
- $Q(6, 3)$ is true.
- $Q(1, 2)$ is false.

- Let $R(x, y, z)$ denote the statement "$x + y = z$".
- $R(1, 2, 3)$ is true.
- $R(3, 4, 5)$ is false.

Quantification

- Let **P(x)** denote the statement “x > 0 **and** x < 10”.
- **P(x)** isn’t a proposition, but P(1), P(2) are all propositions and are true.

- Basically, there are two ways to convert **P(x)** into a proposition.
- Fix x to a certain value, say, 10: P(10)
- Quantify x:
  - “P(x) is true for **all** values of x” (universal quantification).
  - “There **exists** one value of x such that P(x) is true” (existential quantification).

Universal quantifier

- Quantification assumes that the set of possible values of x is well defined, say, the set of positive integers.
- This set of values is called the domain or universe of discourse.

- Universal quantification: "**∀x P(x)**" denotes the proposition "P(x) is true for all values of x in the domain".
- ∀ is often read as "**for all**", "**for every**", or "**for any**".

- E.g., **∀x** (x + 1 > 0) is true.

  ∀x (x - 5 > 0) is false.

Existential quantifier

- Quantification assumes that the set of possible values of x is well defined, say, the set of positive integers.
- This set of values is called the domain or universe of discourse.

- Existential quantification: "$\exists$ **x P(x)**" denotes the proposition "There exists an x in the domain such that P(x) is true".
- $\exists$ is often read as "**there exists**".

- E.g., $\exists$ **x** (x + 1 > 0) is true.

  $\exists$ **x** (x - 5 > 0) is true.

Finite Domain

- Consider a variable x of which the domain contains a fixed number of values, say, 1, 2, 3, 4, and 5.

- **$\forall x\, P(x)$** is equivalent to $P(1) \land P(2) \land P(3) \land P(4) \land P(5)$.

- **$\exists\, x\, P(x)$** is equivalent to $P(1) \lor P(2) \lor P(3) \lor P(4) \lor P(5)$.

Negation

Is $\neg(\forall x\, P(x))$ equivalent to $\exists x\, \neg P(x)$?
I.e., $\neg(\forall x\, P(x)) \Leftrightarrow \exists x\, \neg P(x)$ is true or false?

- YES.
- Suppose "$\neg(\forall x\, P(x))$" is true.
  $\forall x\, P(x)$ is false.
  There exists x such that P(x) is false.
  "$\exists x\, \neg P(x)$" is true.

- Suppose "$\neg(\forall x\, P(x))$" is false.
  $\forall x\, P(x)$ is true.
  "$\exists x\, \neg P(x)$" is false.

Similarly, $\neg(\exists x\, P(x))$ equivalent to $\forall x\, \neg P(x)$.

Existential quantifier in Python

$\exists x\ (x + 1 > 0)$

- To evaluate the above proposition with existential quantifier in Python, we need to assume a finite and reasonably small domain (for the set of positive integers), e.g., **U = [1, 2, 3, 4, 5, 6]**.
- Then, we can use **list comprehension**, as follows:

unit01.ipynb

```
def example1():
    U = [1, 2, 3, 4, 5, 6]
    result = [x for x in U if x + 1 > 0]
    print("result =", result)
    print('"There exists x such that x + 1 > 0" =',
          len(result) != 0)
example1()
```

Universal quantifier in Python

$\forall \mathbf{x}\ (x - 5 > 0)$

- We use negation to change it to a proposition with existential quantifier:
  $\neg\ \forall \mathbf{x}\ (x - 5 > 0)$
  $\equiv \exists\ \mathbf{x} \neg\ (x - 5 > 0)$
  $\equiv \exists\ \mathbf{x}\ (x - 5 \leq 0)$

unit01.ipynb

```
def example2():
    U = [1, 2, 3, 4, 5, 6]
    result = [x for x in U if x - 5 <= 0]
    print("result =", result)
    print('"For all x, x - 5 > 0" =', len(result) == 0)

example2()
```

- If **result** is not empty, each such item is a **counterexample**.

Multiple quantifiers

- $\forall x \, \forall y \, P(x, y)$

  True: P(x, y) is true for all x, y.

  False: There is an x and a y such that P(x, y) is false.

- $\forall x \, \exists y \, P(x, y)$

  True: For every (all) x, there is a y such that P(x, y) is true.

  False: There is an x such that for every y, P(x, y) is false.

- $\exists x \, \forall y \, P(x, y)$

  True: There is an x such that for every y, P(x, y) is true.

  False: For every (all) x, there is a y such that P(x, y) is false.

- $\exists x \, \exists y \, P(x, y)$

  True: There is an x and a y such that P(x, y) is true.

  False: P(x, y) is false for all x, y.

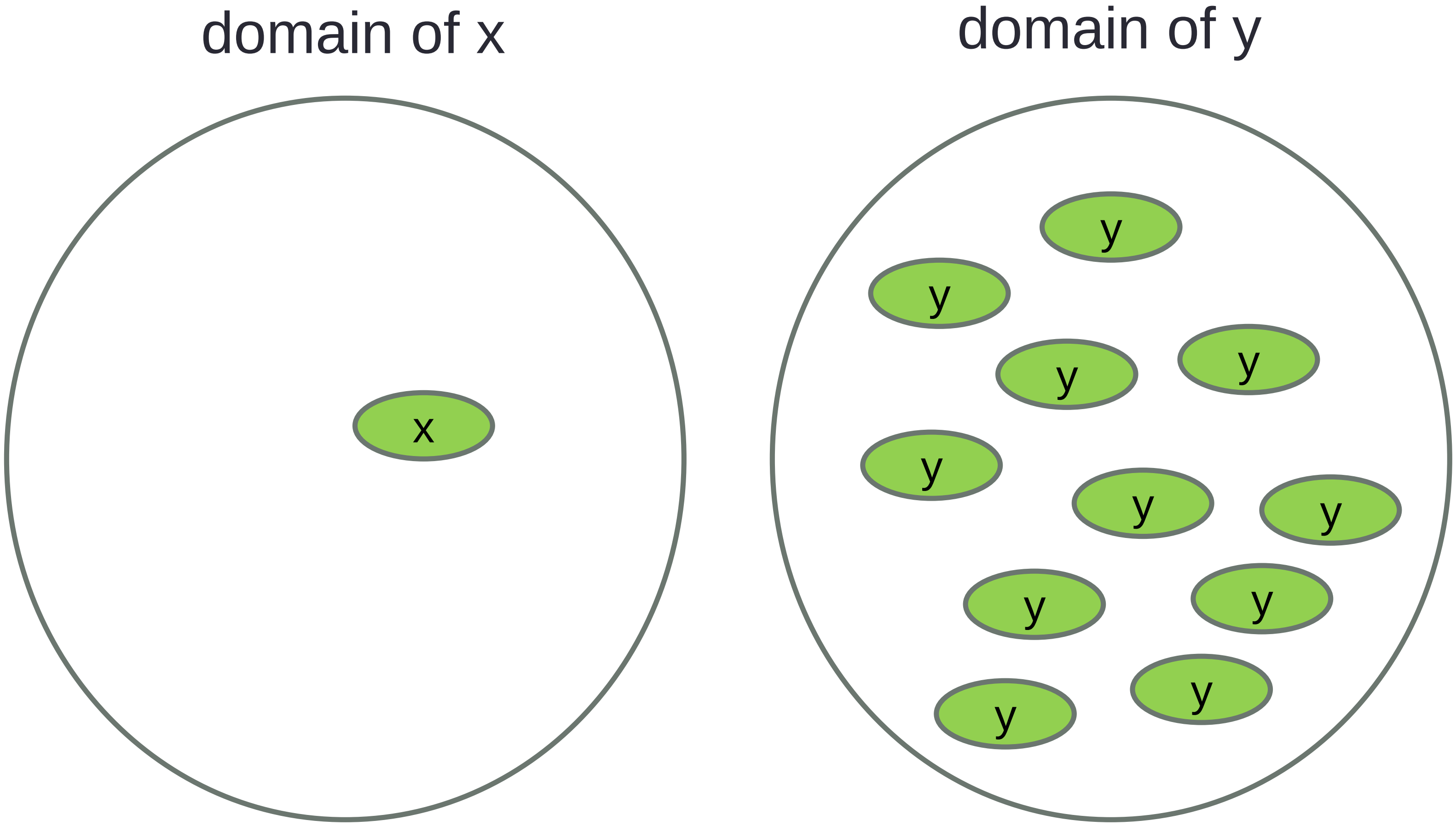
Examples

- Let $P(x, y)$ denote $x + y = 3$.
- Assume x and y are chosen from a domain with a fixed number of values {-1, 0, 1, 2, 3, 4}.

True or False?

- $\forall x\ \forall y\ P(x, y)$
- $\forall x\ \exists\ y\ P(x, y)$
- $\exists\ x\ \forall y\ P(x, y)$
- $\exists\ x\ \exists\ y\ P(x, y)$

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



Multiple quantifiers in Python: mqex1

$\forall x \, \forall y \, (x+y = 3)$

- We use negation on it:

  $\neg \forall x \, \forall y \, (x+y = 3)$

  $\equiv \exists x \, \neg \forall y \, (x+y = 3)$

  $\equiv \exists x \, \exists y \, \neg (x+y = 3)$

  $\equiv \exists x \, \exists y \, (x+y \neq 3)$

unit01.ipynb

```
def mqex1():
    U = [-1, 0, 1, 2, 3, 4]
    result = [(x, y) for x in U for y in U if x + y != 3]
    print("result =", result)
    print('"For all x, for all y, x+y = 3" =', len(result) == 0)

mqex1()
```

Multiple quantifiers in Python: mqex1

∀x ∀y (x+y = 3)

- Output of mqex1():

```
result = [(-1, -1), (-1, 0), (-1, 1), (-1, 2), (-1, 3), (0, -1),
(0, 0), (0, 1), (0, 2), (0, 4), (1, -1), (1, 0), (1, 1), (1, 3),
(1, 4), (2, -1), (2, 0), (2, 2), (2, 3), (2, 4), (3, -1), (3, 1),
(3, 2), (3, 3), (3, 4), (4, 0), (4, 1), (4, 2), (4, 3), (4, 4)]
"For all x, for all y, x+y = 3" = False
```

- Therefore, **∀x ∀y** (x+y = 3) ≡ false as we have the *counterexample* that when x = -1 and y = -1, x+y = -2 ≠ 3.

Multiple quantifiers in Python: mqex2

$\forall x \, \exists y \, (x+y = 3)$

- We use **negation** on it:

$\neg \forall x \, \exists y \, (x+y = 3)$

$\equiv \exists x \, \neg \exists y \, (x+y = 3)$

unit01.ipynb

```
def mqex2():
    U = [-1, 0, 1, 2, 3, 4]
    X = [x for x in U if len([y for y in U if x+y == 3]) == 0]
    print("X =", X)
    print("result =", [(x, [y for y in U if x+y == 3]) for x in U])
    print('"For all x, there exists y s.t. x+y = 3" =',
          len(X) == 0)

mqex2()
```

Multiple quantifiers in Python: mqex2

$\forall$**x** $\exists$ **y** (x+y = 3)

- Output of mqex2():

```
X = []
result = [(-1, [4]), (0, [3]), (1, [2]), (2, [1]), (3, [0]), (4, [-1])]
"For all x, there exists y s.t. x+y = 3" = True
```

Proof.

- For any integer $-1 \leq x \leq 4$,
  $x + y = 3 \Rightarrow y = 3 - x$
  $\Rightarrow$ y is an integer such that $3\text{-}4 \leq y \leq 3\text{-}(-1)$, i.e., $-1 \leq y \leq 4$.
- Therefore, $\forall$**x** $\exists$ **y** (x+y = 3) $\equiv$ true.

Multiple quantifiers in Python: mqex3

$\exists$ x $\forall$ y (x+y = 3)

- We use negation on the following predicate:

  $\neg \forall y\ (x+y = 3)$

  $\equiv \exists\ y\ (x+y \neq 3)$

unit01.ipynb

```
def mqex3():
    U = [-1, 0, 1, 2, 3, 4]
    X = [x for x in U if len([y for y in U if x+y != 3]) == 0]
    print("X =", X)
    print("result =", [(x, [y for y in U if x+y != 3])
                       for x in U])
    print('"There exists x s.t. for all y, x+y = 3" =',
          len(X) != 0)

mqex3()
```

Multiple quantifiers in Python: mqex3

$\exists$ **x** $\forall$ **y (x+y = 3)**

- Output of mqex3():

```
X = []
result = [(-1, [-1, 0, 1, 2, 3]), (0, [-1, 0, 1, 2, 4]), (1, [-1, 0, 1,
3, 4]), (2, [-1, 0, 2, 3, 4]), (3, [-1, 1, 2, 3, 4]), (4, [0, 1, 2, 3,
4])]
"There exists x s.t. for all y, x+y = 3" = False
```

Proof.

- For any integer $-1 \leq x \leq 4$,
  we can set y = x such that x+y = 2x is an even number and is not equal to the odd number 3.
  Therefore, $\exists$ **x** $\forall$ **y (x+y = 3)** $\equiv$ false.

Multiple quantifiers in Python: mqex4

$\exists$ **x** $\exists$ **y** (x+y = 3)

unit01.ipynb

```
def mqex4():
    U = [-1, 0, 1, 2, 3, 4]
    result = [(x, y) for x in U for y in U if x + y == 3]
    print("result =", result)
    print('"There exists x and y s.t. x+y = 3" =',
          len(result) != 0)

mqex4()
```

Multiple quantifiers in Python: mqex4

$\exists$ **x** $\exists$ **y** (x+y = 3)

- Output of mqex4():

```
result = [(-1, 4), (0, 3), (1, 2), (2, 1), (3, 0), (4, -1)]
"There exists x and y s.t. x+y = 3" = True
```

Proof.

- When x=-1, y=4, x+y = 3.
- Therefore, $\exists$ **x** $\exists$ **y** (x+y = 3) $\equiv$ true.

Examples

- Let $P(x, y)$ denote $x + y > 3$.
- Assume x and y are chosen **from the set of integers**.

True or False?

- $\forall x \, \forall y \, P(x, y)$
- $\forall x \, \exists y \, P(x, y)$
- $\exists x \, \forall y \, P(x, y)$

Translating sentences into logical expressions

- Let **B**(x, y) denote the statement "y is a friend of x".
- Which statements have the meaning "Everyone has exactly one friend"?

- $\forall x \, \exists y \; \mathbf{B}(x, y)$
- $\forall x \, \exists y \; (\mathbf{B}(x, y) \wedge \forall z \, \neg \mathbf{B}(x, z))$
- $\forall x \, \exists y \; (\mathbf{B}(x, y) \wedge \forall z \, [z=y \vee \neg \mathbf{B}(x, z)])$
- $\forall x \, \exists y \; (\mathbf{B}(x, y) \wedge \forall z \, [z \neq y \rightarrow \neg \mathbf{B}(x, z)])$
- $\forall x \, \exists y \, \forall z \; (\mathbf{B}(x, y) \wedge [z \neq y \rightarrow \neg \mathbf{B}(x, z)])$