



# CAB203: Discrete Structures Summary

*Author not responsible for consequences caused by any inaccuracies.*

## Markdown Setup

Open in Visual Studio Code and install Markdown Preview Enhanced:

<https://marketplace.visualstudio.com/items?itemName=shd101wyy.markdown-preview-enhanced>

## Typesetting Powered by K<sup>A</sup>T<sub>E</sub>X

K<sup>A</sup>T<sub>E</sub>X documentation: <https://katex.org/>

which uses L<sup>A</sup>T<sub>E</sub>X-like syntax to display mathematical symbols

## Axioms

### Natural Numbers

If  $x, y$  and  $z$  are natural numbers:

1. 0 is a natural number
2.  $x = x$
3. if  $x = y$  then  $y = x$
4. if  $x = y$  and  $y = z$  then  $x = z$
5. if  $x = w$  then  $w$  is a natural number
6.  $S(y)$  is a natural number
7.  $S(x) = S(y)$  then  $x = y$

8.  $S(x) = 0$  is always false

## Other Axioms

WIP

## Equivalence

- **Equals:**  $=$
- **Greater than:**  $>$
- **Greater than or equal to:**  $\geq$
- **Less than:**  $<$
- **Less than or equal to:**  $\leq$
- **Not equal to:**  $\neq$

## Division

- $a \mid b$
- $a$  divides  $b$
- $b$  is divisible by  $a$
- there exists some integer  $c$  such that  $ac = b$

## Definitions

May either replace term with its definition, or replace definition with its term:

- the term  $2 \cdot 3 = 6$  may be replaced by  $2 \mid 6$ , while 3 becomes  $c$
- the definition  $2 \mid$   
6 may be replaced with "there is some integer  $c$  such that  $2c = 6$ "

# Parity

Integers have one of two *parities*:

- $x$  is *even* means  $2 \mid x$
- $x$  is *odd* means  $2 \mid (x - 1)$

Properties:

- $\text{even} \pm \text{even} = \text{even}$
- $\text{even} \pm \text{odd} = \text{odd}$
- $\text{odd} \pm \text{odd} = \text{even}$

Therefore, two numbers have same parity if difference is even.

## Clock Arithmetic

- $10 \text{ o'clock} + 5h = 3 \text{ o'clock}$
- the *o'clock* remains unaffected by multiples of  $12h$

## Modular Arithmetic

*Modular arithmetic* is an abstraction of parity and clock arithmetic.

- parity is *arithmetic mod 2*
- clocks use *arithmetic mod 12*
- generally, can have *arithmetic mod n* for any positive integer  $n$

Modular arithmetic extends upon integers by adding a new relation (modular equivalence)

## Modular Equivalence

Modular arithmetic works by replacing equality with *modular equivalence*, also called *modular congruence*:

if  $n \mid (a - b)$ , then  $a$  and  $b$  are *equivalent modulo n*, such that  $a \equiv b \pmod{n}$

if  $a \equiv b \pmod{n}$  and  $c \equiv d \pmod{n}$ , then:

- $a + c \equiv b + d$
- $a - c \equiv b - d \pmod{n}$
- $ac \equiv bd \pmod{n}$

## Mod Operator

$a \bmod n$  is the smallest non-negative  $b$  such that  $a \equiv b \pmod{n}$

- Equivalently,  $a \bmod n$  is the remainder from  $\frac{a}{n}$
- *example* :  $17 \bmod 4 = 1$  because  $17 = 4(4) + 1$

Python:

```
print(17 % 4) # Prints 1 to the console  
# 1
```

## Modulo Lemma

A *lemma* is a short statement that is true in mathematical theory, derived from its axioms. We can show that it is true by using a *proof* that shows the steps from the axioms to the statement. Other axioms with existing proofs may be used in the proof.

## Proof of Lemma

Let integers  $a$  and  $b$  be given such that  $a \bmod b = 0$ .

Then from the definition of the mod operator:

$$a \equiv 0 \pmod{b}$$

From the definition of modular equivalence:

$$b \mid (a - 0)$$

From a well known lemma observe that  $a - 0 = a$  for any integer, so  $b \mid a$ .

This is frequently used to determine divisibility or test if a number is even when programming.

# Exponents

- $a^3$  means  $a \cdot a \cdot a$
- $a^n$  means multiply  $a$  together  $n$  times
  - $a$  is called the base
  - $n$  is called the exponent

## Laws of Exponents

- $(ab)^n = a^n \cdot b^n$
- $a^m \cdot a^n = a^{m+n}$
- $a^{m-n} = \frac{a^m}{a^n}$  (when  $a \neq 0$ )
- $a^{-n} = \frac{1}{a^n}$  (when  $a \neq 0$ )
- $a^0 = 1$
- $(a^m)^n = a^{m \cdot n}$

## Exponents in Computer Science

- **kilo:**  $2^{10}$
- **mega:**  $2^{20} = (2^{10})^2$
- **giga:**  $2^{30} = (2^{10})^3$
- **tera:**  $2^{40} = (2^{10})^4$
- **peta:**  $2^{50} = (2^{10})^5$
- **exa:**  $2^{60} = (2^{10})^6$

For example, one kilobit is  $1024 = 2^{10}$  bits.

- $2^3 = 8$  bits in a byte
- $2^{10} = 1024$  bytes in a kilobyte
- $2^{10} \cdot 2^3 = 2^{10+3}$  bits in a kilobyte
- $32 = 2^5$  or  $64 = 2^6$  bit processors
- $256 = 2^8 = 2^{2^3}$  possible 8-bit characters

## Logarithms

- logarithms are the *inverse* of exponents
- if  $n = \log_a x$  then  $a^n = x$
- so  $\log_a$  tells what exponent is needed to make  $x$  from  $a$  :

$$a^{\log_a x} = x$$

## Laws of Logarithms

- $\log_a 1 = 0$
- $\log_a a = 1$
- $\log_a (x \cdot y) = \log_a x + \log_a y$
- $\log_a x^y = y \log_a x$
- $\log_a \frac{1}{y} = -\log_a y$
- $\log_a \frac{x}{y}$
- $\log_b x = (\log_b a) \cdot \log_a x$

## Base Transformation Law

- $\log_a x = \frac{\log_b x}{\log_b a}$
- use base transformation to calculate  $\log_2$ , etc...

## Ceiling and Floor

- **Ceiling, round up:**  $\lceil a \rceil$  is the next integer above  $a$
- **Floor, round down:**  $\lfloor a \rfloor$  is the next integer below  $a$
- $\lceil \log_2 5000 \rceil = 13$  address lines

## Exponent and Logs in Python

```
>>> import math          # Must import math library
>>> math.log2(8)          # log2 means log base 2 and returns a float (decimal)
# 3.0
>>> 2 ** 3               # ** is exponentiation
# 8
>>> math.log2(100) / math.log2(10) # base transformation
# 2.0
```

## Operators

An operator is a mathematical object that transforms other objects:

- $+$  is a binary operator (transforms two objects, ie:  $1 + 2$  into  $3$ )
- $-$  combines two operators
  - As a binary operator:  $1 - 2$
  - As a unary operator:  $-1$  (negative number)

# Bits

## Bit String Notation

- **String:**  $\bar{x}$
- The set of all strings of length  $n$  (aka  $n$ -bit strings) is:  $\{0, 1\}^n$
- All bit strings of all length are members of:  $\{0, 1\}^*$
- The  $j$ th bit in  $\bar{x}$  is:  $\bar{x}_j$  ( $j$  goes from 0 to  $n - 1$ )
- Bit strings are most often counted from the *right*, so the furthest right is:  $\bar{x}_0$
- $2^n$  possible bit strings of length  $n$

## Bit Operations

Two types of bit operations:

- Operations on a single bit or pairs of bits
- operations on bit strings

## NOT

NOT, aka *bit flip*: 0 becomes 1 and vice versa.

$x$	$\sim x$
0	1
1	0

## AND

AND is similar to multiplication.



$x$	$y$	$x \& y$
0	0	0
0	1	0
1	0	0
1	1	1

## OR

OR is similar to addition, yet 2 is condensed to 1.

$x$	$y$	$x \mid y$
0	0	0
0	1	1
1	0	1
1	1	1

## XOR

XOR is also similar to addition, yet 2 is now condensed to 0, like parity.

$x$	$y$	$x \wedge y$
0	0	0
0	1	1
1	0	1
1	1	0

## Bitwise Operations

Bit operations may be applied bitwise to strings of the same length.

if  $\bar{x}$  &  $\bar{y}$  then

$$\bar{z}_j = \bar{x}_j \& \bar{y}_j$$

Operations are performed on pairs of bits.

## Concatenation

if  $\bar{x}$  is an  $n$ -bit string and  $\bar{y}$  is a  $m$ -bit string, then  $\bar{z} = \overline{xy}$  is a  $(n + m)$ -bit string

## Lexicographic Ordering

- 0 before 1
- Compare strings one bit at a time, left to right
- At first bit where strings differ, 0 goes first
- Shorter strings are padded with empty spaces to the right

## ASCII

- 7 bit strings
- 128 characters
- Upper, lower case Latin chars, numbers, punctuation, maths symbols, space, newline, etc
- Special characters BEL, ESC, NUL, etc
- Relational blocks
- Upper vs lower is just one bit
- Letters and numbers ordered lexicographically

## USASCII Code Chart

$b_7 \rightarrow$	$b_6 \rightarrow$	$b_5 \rightarrow$	$\Rightarrow$	$\Rightarrow$	000	001	010	011	100	101	110	111
$b_4 \downarrow$	$b_3 \downarrow$	$b_2 \downarrow$	$b_1 \downarrow$	$r \downarrow c \rightarrow$	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	'	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	\$	4	E	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	F	U	e	u
0	1	1	0	6	ACK	SYN	&	6	G	V	f	v
0	1	1	1	7	BEL	ETB	'	7	H	W	g	w
1	0	0	0	8	BS	CAN	(	8	I	X	h	x
1	0	0	1	9	HT	EM	)	9	J	Y	i	y
1	0	1	0	10	LF	SUB	*	:	K	Z	j	z
1	0	1	1	11	VT	ESC	+	;	L	[	k	{
1	1	0	0	12	FF	FS	,	<	M	\	l	
1	1	0	1	13	CR	GS	-	=	N	]	m	}
1	1	1	0	14	SO	RS	.	>	O	^	n	~
1	1	1	1	15	SI	US	/	?	Q	_	o	END

## UNICODE

- About 137 000 characters
- Most modern and some historic writing systems
- Mathematical symbols, punctuation, emoji, etc
- Multiple encodings for a common set of characters

# UNICODE Encodings

Unicode assigns a code point (a hexadecimal string) for each character. There are several different encodings from code points to bit strings:

- UTF32 uses 32 bits for each character, encoding code points directly
- UTF16 uses one or two 16-bit strings per code point, making it a variable length encoding
- UTF8 uses between one and four 8-bit strings per code point, and is hence also a variable length encoding.
- It is backwards compatible with ASCII for the original 7-bit ASCII character set
  - UTF8 is the most common encoding. Python strings are UTF-8 encoded by default.

## UTF-8 in Python

```
>>> ord('A')    # Convert character to UTF code point
# 65
>>> chr(65)     # Convert UTF code point to character
# 'A'
```

# Numbers

## Binary Representation

Binary numbers are analogous to base-10 notation:

- Starts at position 0, the right-most numeral
- Position  $j$  gets a multiplier of  $2^j$
- Add up all values

Example, 1010, starting from position 0:

- 0 has multiplier of  $2^0 = 1$
- 1 has multiplier of  $2^1 = 2$
- 0 has multiplier of  $2^2 = 4$

- 1 has multiplier of  $2^3 = 8$
- total is 10

## 4-Bit Binary

<i>base - 10</i>	<i>base - 2</i>	<i>base - 10</i>	<i>base - 2</i>
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

## 8-Bit Binary

Storing positive numbers (0...255), as 8-bit strings:

$$\sum_{j=0}^7 2^j \bar{x}_j$$

## Adding Binary Numbers

Adding two 1-bit numbers:

$$0 + 0 = 0 \quad (1)$$

$$1 + 0 = 1 \quad (2)$$

$$0 + 1 = 1 \quad (3)$$

$$1 + 1 = 10 \quad (4)$$

# Bit Operations

Given two 1-bit numbers,  $\bar{x}$  and  $\bar{y}$ , their binary sum is a 2-bit string  $\bar{z}$  where:

$$\bar{z}_0 = \bar{x}_0 \wedge \bar{y}_0 \quad (5)$$

$$\bar{z}_1 = \bar{x}_0 \& \bar{y}_0 \quad (6)$$

For  $n$ -bit binary numbers  $\bar{x}$  and  $\bar{y}$ , the sum in binary  $\bar{z}$  is a  $n + 1$ -bit binary number where:

$$\bar{z}_j = \bar{x}_j \wedge \bar{y}_j \wedge \bar{c}_j \quad (7)$$

$$\bar{c}_{j+1} = (\bar{x}_j \& \bar{y}_j) \mid (\bar{x}_j \& \bar{c}_j) \mid (\bar{y}_j \& \bar{c}_j) \quad (8)$$

The string  $\bar{c}$  is the carry bits (take  $\bar{c}_0$  to be 0). The equation for  $\bar{c}_{j+1}$  says it is 1 when  $\bar{x}_j + \bar{y}_j + \bar{c}_j$  is 2 or 3.

## Negative Numbers

Properties of 2's complement:

- For  $n$  bits, can represent  $-2^{n-1}$  through to  $2^{n-1} - 1$
- Leftmost bit is 1 for negative numbers
- Addition is  $\text{mod } 2^n$
- Positive numbers 0 to  $2^{n-1} - 1$  are unchanged

## 3-Bit 2's Complement

bit string	2's comp interpretation	binary interpretation
000	0	0
001	1	1
010	2	2
011	3	3
100	−4	4
101	−3	5
110	−2	6
111	−1	7

## Hexadecimal

- Base-64 number system - number in position  $j$  gets a multiplier of  $16^j$ .
- Compact method for writing bit strings

## 4-Bit Hex

symbol	bit string	base-10	symbol	bit string	base-10
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	A	1010	10
3	0011	3	B	1011	11
4	0100	4	C	1100	12
5	0101	5	D	1101	13
6	0110	6	E	1110	14
7	0111	7	F	1111	15

# Interpreting Hex

given:

$4F = 4 \rightarrow 0100$  and  $F \rightarrow 1111 = 01001111$

then:

$4F = 01001111$

- Shorter than bit strings
- One hex numeral is always exactly 4 bits
- Easy to work with individual bits

# Python Hex

```
>>> hex(65532)                # Hex string from integer
# '0xffffc'
>>> "The number is {:x}".format(65532) # Alternative method
# 'The number is fffc'
>>> 0xffffc                    # Hex literal integer
# 65532
```

# Scientific Notation

$$\pm a.bc \times 10^e \quad (9)$$

- **Sign:**  $\pm$
- **Significant digits:**  $a.bc$
- **Exponent:**  $e$
- **Base:** 10

The base is always shared across the significant digits

# Scientific Notation in Base 2

- Significant digits are bits
- Exponent is written in binary



$$1.1010 \times 2^{10} \quad (10)$$

## Floting Point Numbers

Computers represent scientific notation as floating point numbers, encoding in binary:

- Significant digits
- Exponent
- Sign (positive or negative)
- With base 2

## IEEE Half-Precision

IEEE 754 floating point standard for 16 bits:

$$\begin{array}{c} s \quad e \quad f \\ \underbrace{0} \quad \underbrace{10101} \quad \underbrace{1010101010} \end{array} \quad (11)$$

- **Sign:**  $s$  (1-bit)
- **Exponent:**  $e$  (5 bits)
- **Significant digits:**  $f$  (10 bits dropping leading 1)

## Python Numers

Basic number types:

- `int`
  - Arbitrary length integers
  - Special base-2<sup>30</sup>
- `float`
  - IEEE 754 double precision
    - 64-bit floating point

```
>>> 9/2      # Regular division always returns float
# 4.5
>>> 9//2     # Floor division returns integer
# 4
```

# Recursion

## Recursive Definitions

Factorial function on  $\mathbb{N}$  defined as:

$$n! = \prod_{j=1}^n j = 1 \cdot 2 \cdots (n-1) \cdot n \quad (12)$$

Also  $n!$  recursively:

$$n! = \begin{cases} 1 & : n = 1 \\ n(n-1)! & : n > 1 \end{cases} \quad (13)$$

Two main parts:

- **Base case:** evaluated without reference to object
  - Required, though potentially multiple
- **Recursive case:** refer back to object definition
  - More complex than base

## Fibonacci Sequence

Fibonacci sequence is classic example of recursion:

$$f(n) = \begin{cases} 1 & : n = 1 \\ 1 & : n = 2 \\ f(n-1) + f(n-2) & : n > 2 \end{cases} \quad (14)$$

## Fibonacci in Python

```
def F(n):  
    if n == 1: return 1  
    elif n == 2: return 1  
    else: return F(n-1) + F(n-2)
```

# Arithmetic Expression

Programming languages often expressed as multiple types in recursion:

$$EXPR := \begin{cases} VALUE \\ EXPR \text{ “ + ” } VALUE \\ EXPR \text{ “ - ” } VALUE \end{cases} \quad (15)$$

$$VALUE := \begin{cases} CONSTANT \\ VARIABLE \end{cases} \quad (16)$$

## Propositional Logic

- Propositions are true or false statements
- Logical connectives use propositions to build larger ones
- $p$  and  $q$  often represent propositions

## Atomic and Compound Propositions

- **Atomic:**    "It is raining"
- **Compound:**    "It is raining and cloudy"

## Logical Operators

- **NOT, negates truth:**     $\neg$
- **AND, requires both:**     $\wedge$
- **OR, allows either:**     $\vee$
- **XOR, requires either:**     $\oplus$
- **IF..THEN:**     $\rightarrow$
- **IF AND ONLY IF:**     $\leftrightarrow$

## IF..THEN

- $p \rightarrow q$  means  $q$  must be true whenever  $p$  is, regardless of  $p$

- When  $p$  is false,  $p \rightarrow q$  is always true
- $(p \rightarrow q) = (\text{if } p \text{ then } q) = (p \text{ implies } q)$

$p$	$q$	$p \rightarrow q$
$T$	$T$	$T$
$T$	$F$	$F$
$F$	$T$	$T$
$F$	$F$	$T$

## IF AND ONLY IF

- $p \leftrightarrow q$  means both must have the same truth value
- $(p \leftrightarrow q) = ((p \rightarrow q) \wedge (q \rightarrow p)) = (p \text{ if and only if } q)$

$p$	$q$	$p \leftrightarrow q$
$T$	$T$	$T$
$T$	$F$	$F$
$F$	$T$	$F$
$F$	$F$	$T$

## Formulas

*Boolean formulas* are strings of symbols to build compound propositions.

### Formula rules (listed by precedence):

- $T$ ,  $F$  and lower case letters are all formulas
- If  $A$  and  $B$  are formulas then so are:
  - $(A)$
  - $\neg A$
  - $A \oplus B$
  - $A \wedge B$

- $A \vee B$
- $A \rightarrow B$
- $A \leftrightarrow B$
- No other strings are formulas

## Formula Truth Value

- Fill in truth table
- Evaluate logical connectives from innermost parentheses outwards

When  $p = T$  and  $q = F$ :

$$\begin{aligned}
 (p \vee q) \rightarrow (q \oplus p) &= (T \vee F) \rightarrow (F \oplus T) \\
 &= T \rightarrow T \\
 &= T
 \end{aligned} \tag{17}$$

## Formula Classification

- **Tautologies:** always  $T$
- **Contradictions:** always  $F$
- **Contingent formulas:**  $T$  OR  $F$  depending on variables
- **Satisfiable formulas:** *tautologies* OR *contingent formulas*

## Logical Equivalence

$$A \rightarrow B \equiv \neg A \vee B \tag{18}$$

$A \equiv B$  is the same as stating: “ $A \leftrightarrow B$  is a *tautology*”

$$\begin{aligned}
 A \rightarrow B &\equiv \neg A \wedge B \\
 &\equiv B \wedge \neg A \\
 &\equiv \neg(\neg B) \wedge \neg A \\
 &\equiv \neg B \rightarrow \neg A
 \end{aligned} \tag{19}$$

# Set Theory

- **Set:**  $S$
- **In set:**  $\in$
- **Not in set:**  $\notin$

**Set listing:**

$$SMALLPRIMES = \{ 1, 2, 3, 5, 7 \} \quad (20)$$

**Implied pattern:** (*bad practice*)

$$EVENS = \{ 2, 4, 6, 8, \dots \} \quad (21)$$

**Setbuilder notation:** “set comprehension”

Subset of elements, that match a condition

$$\{ x \in S : \phi(x) \} \quad (22)$$

$$SQUARES = \{ x \in \mathbb{Z} : x = y^2 \text{ for some } y \in \mathbb{Z} \} \quad (23)$$

**Setbuilder notation:** “replacement”

Apply a theory to each member and collect results

$$\{ f(x) : x \in S \} \quad (24)$$

$$SQUARES = \{ x^2 : x \in \mathbb{Z} \} \quad (25)$$

## Set Equality

$S = T$  when:

- Every element  $x \in S$  is in  $T$
- Every element  $x \in T$  is in  $S$
- Regardless of order, repetition, and representation:

$$\{1, 2, 3\} = \{3, 2, 1\} \quad (26)$$

$$\{1, 1, 1\} = \{1\} \quad (27)$$

$$\{x \in \mathbb{Z} : x^2 = 4\} = \{2, -2\} \quad (28)$$

## Set Size

$$|\{1, 2, 3\}| = 3 \quad (29)$$

$$|\{1, 1, 1\}| = 1 \quad (30)$$

## Sub Sets

- **Subset:** every  $x \in A$  is in  $B$  :  $A \subseteq B$
- **Proper subset:**  $A$  is subset of, yet not equal to  $B$  :  $A \subset B$
- **Equality through subsets:**  $S \subseteq T$  &  $T \subseteq S$

Subset equivalence:

$$A \subset B \equiv A \subseteq B \wedge A \neq B \quad (31)$$

The set of all subsets of a set  $S$  is called the *power set* of  $S$ :

$$P(\{1, 2\}) = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\} \quad (32)$$

## Set Operations

**Union:** everything in either

Must define  $A \cup B$  to be the smallest set, such that  $A \subseteq S$  &  $B \subseteq S$

$$A \cup B = \{x : x \in A \vee x \in B\} \quad (33)$$

**Intersection:** everything in *both* sides

$$A \cap B = \{x \in A : x \in B\} \quad (34)$$

**Difference:** remove items in one set from the other

$$A \setminus B = \{x \in A : x \notin B\} \quad (35)$$

## Universe Sets

- The universe  $U$  is the set containing all elements of concern
- Enables definition of complements:

$$\overline{S} = \{x \in U : x \notin S\} \quad (36)$$

Set comprehensions without specifying the set, as with  $U = \mathbb{Z}^+$ :

$$COMPOSITES = \overline{PRIMES} \quad (37)$$

$$EVENS = \{x : x = 2y\} \quad (38)$$

$$ODDS = \overline{EVENS} \quad (39)$$

## Characteristic Vectors

Universes of manageable size may be represented as bit strings (characteristic vectors):

- Let  $n = |U|$
- Number elements like so,  $U = \{e_1, e_2, \dots, e_n\}$
- $XS = XS_1XS_2 \dots XS_n$  where:

$$XS_j = \begin{cases} 0 & e_j \notin S \\ 1 & e_j \in S \end{cases} \quad (40)$$

- Example:  $U = \{1, 2, 3\}$  then  $X_{\{1,3\}} = 101$ ,  $X_{\{2\}} = 010$



- Set operations such as  $\cup$  and  $\cap$  become bitwise operators

# Python Sets

```

>>> S = {1,2,3}; T = {1,3,5}      # Braces define sets
>>> S.add(4); print(S)            # Sets are mutable
# {1, 2, 3, 4}
>>> S.remove(4); print(S)
# {1, 2, 3}

>>> 1 in S                        # Testing membership
# True
>>> 4 in S
# False

>>> S.issubset({1,2,3,4,5})      # Testing subset
# True
>>> S <= {1,2,3,4,5}            # Alternative subset test
# True

>>> S.union(T)                   # Using union
# {1, 2, 3, 5}
>>> S | T                        # Alternative union
# {1, 2, 3, 5}
>>> S.union(T, {8,9}, {10,11})  # Multiple union
# {1, 2, 3, 5, 8, 9, 10, 11}
>>> someSets = [S, T, {8,9}, {10,11}]
>>> set.union(*someSets)        # Splat operator
# {1, 2, 3, 5, 8, 9, 10, 11}

>>> S.intersection(T)           # Splat and multi-sets would also work
# {1, 3}
>>> S & T                         # Alternative intersection
# {1, 3}

>>> S - T                        # Testing difference
# {2}

>>> S.isdisjoint(T)             # is S & T empty?
# False

>>> len(S)                      # Size of S
# 3

# Setbuilder notation combines of comprehension and replacement:
>>> S = {0, 2, 4, 6, 8}
>>> def p(x): return x % 2 == 0
...
>>> def p(x): return x * 5
...

```

```
>>> { f(x) for x in S if p(x) }  
# {0, 40, 10, 20, 30}  
>>> { s * 5 for s in range(0, 10) if s % 2 == 0 }  
# {0, 40, 10, 20, 30}
```

## Zermelo-Fraenkel Set Theory

ZF set theory, seven axioms to define set behaviour:

1. Two sets containing same elements are equal
2. Every set  $S$  other than  $\emptyset$  contains at least one element  $y$ , also  $S$  and  $y$  are disjoint
3. If  $S$  is a set and  $\phi(x)$  is a formula, then there is a set that contains exactly the elements of  $S$  that satisfies  $\phi(x)$
4. If  $S_1, S_2, \dots$  are sets, then there is a set that contains all of the elements of every  $S_j$ 
  - *Allows unions*
5. If  $S$  is a set and  $f(x)$  is a function, then there is a set that contains  $f(x)$  for every  $x \in S$ 
  - *Allows replacements such as:  $\{ f(x) : x \in S \}$*
6. Given  $S_0 = \emptyset$  and  $S_j = S_{j-1}$ , there is a set that contains every  $S_j$
7. For a set  $S$ , there is a set containing every possible subset of  $S$

- *Allows power sets*

## Replacements From ZF Axioms

ZF replacements from axioms 1, 3, and 5:

- Start with set  $S$  and function  $f(x)$
- Use axiom 5 to obtain set  $A$  containing  $f(x)$  whenever  $x \in S$
- Create formula  $\phi(y)$  so  $x \in S$ , such that  $f(x) = y$
- Use axiom 3 to obtain set  $A'$  containing every  $y \in A$  such that  $\phi(y)$  is true

$A'$  is the resulting set.

## $\mathbb{Z}_{\geq 0}$ From Set Theory

Peano's axioms to construct set of non-negative integers:

- What is 0
- A successor function  $S(\cdot)$  that takes a number to the next one:  
 $S(1) = 2, S(2) = 3 \dots$   
 Define empty set,  $\emptyset$ , as 0 and define a successor function by  $S(n) = n \cup \{n\}$ :
- “0” :=  $\emptyset = \{\}$
- “1” :=  $S(\text{“0”}) = \text{“0”} \cup \{\emptyset\} = \{\{\}\}$
- “2” :=  $S(\text{“1”}) = \text{“1”} \cup \{\text{“1”}\} = \{\emptyset, \{\emptyset\}\} = \{\{\}, \{\{\}\}\}$
- $\vdots$
- $S(n) = n \cup \{n\}$

## Syllogisms

Syllogisms are deductive reasoning upon sets:

- Start with *premises* (statements), taken to be true
- Apply valid form of argument
- Draw conclusion
- If *premises* are true, then impossible for conclusion to be false:

$$\begin{array}{c}
\text{All humans are mortal } (\textit{major premise}) \\
\text{Socrates is human } (\textit{minor premise}) \\
\hline
\text{Socrates is mortal } (\textit{conclusion})
\end{array}
\tag{41}$$

## Set Theoric Syllogisms

$$\begin{array}{c}
HUMANS \subseteq MORTALS \\
s \in HUMANS \\
\hline
s \in MORTALS
\end{array}
\tag{42}$$

## Syllogism Types

Abstracted to *syllogism type*:

$$\begin{array}{c}
A \subseteq B \\
x \in A \\
\hline
x \in B
\end{array}
\tag{43}$$

This is *valid* type, so works for any  $A, B, x$  provided the *premises* are true

24 valid syllogism types in total, such as:

$$\begin{array}{c}
\text{All trees are plants } (A \subseteq B) \\
\text{Some trees are tall } (A \cap C \neq \emptyset) \\
\hline
\text{Some plants are tall } (B \cap C \neq \emptyset)
\end{array}
\tag{44}$$

$$\begin{array}{c}
\text{All cats are mammals } (A \subseteq B \text{ and } A \neq \emptyset) \\
\text{All cats are carnivores } (A \subseteq C) \\
\hline
\text{Some mammals are carnivores } (B \cap C \neq \emptyset)
\end{array}
\tag{45}$$

# Logical Implication

- From  $A \wedge B$ , can conclude  $A$
- From  $(A \rightarrow B) \wedge A$ , can conclude  $B$
- **Implication:**  $A \models B$

If from  $A$ , can conclude  $B$ , then  $A \models B$ , then  $A \rightarrow B$  is a tautology

Whenever  $A$  is true, so is  $A \vee B$ , thus  $A \models A \vee B$ :

$A$	$B$	$A \vee B$	
$T$	$T$	$T$	$\leftarrow$
$T$	$F$	$T$	$\leftarrow$
$F$	$T$	$T$	<i>(ignore)</i>
$F$	$F$	$F$	

Given  $A \models A \wedge B$ , then  $A \rightarrow A \vee B$  is a tautology

$A$	$B$	$A \vee B$	$A \rightarrow A \vee B$
$T$	$T$	$T$	$T$
$T$	$F$	$T$	$T$
$F$	$T$	$T$	$T$
$F$	$F$	$F$	$T$

## Implication Substitutions

- Given  $A \wedge B \models A$ , if just  $A \wedge B$  is true, then may replace with  $A$
- From “it is cloudy and raining”, can conclude “it is raining”

## Proofs

Given:

- Socrates is mortal or Socrates is not human
- Socrates is human

...then conclude:

- Socrates is not human or Socrates is mortal
  - *equivalence*
- Socrates is mortal
  - *logical implication*

A proof is a list of formulas starting with premises and every formula must be:

- Logically equivalent to a formula above
- Logically implied by a formula above
- The *AND* of some formulas above
- Logically implied by the *AND*

A proof produces  $P \models Q$ , where:

- $P$  is the *AND* of all premises
- $Q$  is the last line of the proof

Proof says: assuming all premises are true, the conclusion is also true

Socrates proof:  $(M \vee \neg H) \wedge H \models M$  :

1	$M \vee \neg H$	premise	
2	$H$	premise	
3	$\neg H \vee M$	equivalent to line 1 using $A \vee B \equiv B \vee A$	
4	$\neg\neg H$	equivalent to line 1 using $A \equiv \neg\neg A$	
5	$M$	implication of line 3 <i>AND</i> 4 using	(46)
		$(A \vee B) \wedge \neg A \models B$ , with $A \models \neg H$ <i>AND</i> $B = M$	



# Predicate Logic

## Parameters and Predicates

Generalise propositions by allowing *parameters*:

$$A(x) = x \text{ is a cat} \quad (47)$$

$$B(x, y) = x \text{ and } y \text{ have the same birthday} \quad (48)$$

$$C(x, y) = x = y + 1 \quad (49)$$

- Parameters allow generic references to propositions that share a common form and meaning
- Predicates are propositions with one or more variables

Form complex predicates from smaller ones:

- $A(x) \wedge B(x)$  understood that  $x$  is the same for both
- $A(x) \vee B(y)$  can have different parameters
- $A(x) \rightarrow B(x)$
- $A((x) \rightarrow B(y)) \wedge A(x)$

To evaluate truths, must fill parameters:

- Suppose  $A(x)$  is  $x^2 = 1$ .
  - Then  $A(1)$  is True, but  $A(2)$  is False.
- Suppose  $A(x)$  is  $x$  is a flower  $\rightarrow x$  smells nice.
  - Then  $A(\text{rose})$  is True, but  $A(\text{rafflesia})$  is False.
- Suppose  $A(x)$  is if  $x$  is human then  $x$  is mortal.
  - Then  $A(\text{Socrates})$  is True

Predicates with all variables defined, become regular propositions with truth values

# Quantifiers

Quantifiers are symbols that refer to parameters within predicates:

- **Existential quantification, *there exists*:**  $\exists$
- **Universal quantification, *for all*:**  $\forall$

Quantifiers allow propositions out of predicates without parameter values

## Existential Quantification

$$\begin{aligned} & \exists x \in \mathbb{Z}(x^2 = 4) \\ & \text{there exists an } x \text{ from } \mathbb{Z} \text{ such that } x^2 = 4 \end{aligned} \tag{50}$$

$$\begin{aligned} & \exists x \in S p(x) \\ & \text{there exists an } x \text{ in } S \text{ such that } p(x) \text{ is True} \end{aligned} \tag{51}$$

$$\begin{aligned} & \exists x \in ANIMALS(x \text{ is a fish}) \\ & \text{there exists an } x \text{ in } ANIMALS \text{ such that } x \text{ is a fish} \end{aligned} \tag{52}$$

$$\begin{aligned} & \exists x \in \mathbb{R}(x \in \mathbb{Z}) \\ & \text{there exists some real number } x \text{ such that } x \text{ is an integer} \end{aligned} \tag{53}$$

## Universal Quantification

$$\forall x \in \mathbb{Z} (x^2)$$

for every  $x$  in  $\mathbb{Z}$ ,  $x^2$  is non-negative (54)

$$\forall x \in S p(x)$$

for all  $x$  from  $S$ ,  $p(x)$  is True (55)

$$\forall x \in ANIMALS (x \text{ is a fish})$$

for all  $x$  in  $ANIMALS$ ,  $x$  is a fish (56)

$$\forall x \in \mathbb{Z} (x \in \mathbb{R})$$

all integers are real numbers (57)

## Quantify Over Sets

Explicit set specification, to quantify over  $S$ :

$$\forall x \in S p(x)$$

(58)

Implicit set specification, to quantify over a set, *out of context*:

$$\forall x p(x)$$

(59)

## Parameters in Predicates

Given  $\exists x p(x)$

- The  $x$  is filled in by the  $\exists x$ , so no values allowed
- Either:
  - there exists an  $x$  that makes  $p(x)$  True (making  $\exists x p(x)$  True)
  - or there does not (making it False)

## Free Parameters

Given  $A(y) = \exists x p(x, y)$ :

- The parameter  $x$  is quantified over, so no values allowed

- The parameter  $y$  is *not* quantified over, so values *are* allowed
- Truth value of  $A(y)$  depends on value of  $y$
- Here  $y$  is a free parameter
- When no free parameters, predicate is *fully quantified*

## Fully Quantified Truth Values

Checking all values of fully quantified finite set:

$$\frac{\forall x \in \{0, 1\}(x^2 = x)}{0^2 = 0}$$

$$1^2 = 1$$

*thus, True* (60)

## Existential Truth Values

Ensuring only one value works for existential quantifiers to be True:

$$\frac{\exists x \in \{0, 1\}(x^2 = 1)}{1^2 = 1}$$

*thus, True* (61)

Yet requiring all values to be tested to prove False:

$$\frac{\exists x \in \{0, 1\}(x^2 = 2)}{0^2 \neq 2}$$

$$1^2 \neq 2$$

*thus, True* (62)

Both of these cases are the opposite when dealing with universal quantifiers

## Existential Quantifiers Over Infinite Sets

$$\exists x \in \mathbb{Z}(x^2 = -1) \quad (63)$$

...is False, as for every  $x \in \mathbb{Z}$ ,  $x^2 \geq 0$ , and hence  $x^2 \neq -1$

## Order of Quantifiers

Order impacts multiple quantifiers as with universe  $\mathbb{Z}$ :

$$\forall y \exists x (x + y = 0) \quad (64)$$

True, as can use  $x = -y$

$$\exists x \forall y (x + y = 0) \quad (65)$$

False, as for any  $x$ , can use  $y = x + 1$ ,

so  $x + y = 1 \neq 0$

## IF.. THEN Quantified

Let  $h(x)$  be  $x$  is human and  $m(x)$  be  $x$  is mortal, then:

$$\forall x \in BEINGS (h(x) \rightarrow m(x)) \quad (66)$$

## Necessary and Sufficient Conditions

$$\text{Given } \forall x (p(x \rightarrow q(x))) : \quad (67)$$

$p(x)$  is a *sufficient* condition for  $q(x)$

$q(x)$  is a *necessary* condition for  $p(x)$

$$\text{Given } \forall x (p(x \leftrightarrow q(x))) : \quad (68)$$

$p(x)$  is *necessary* and *sufficient* for  $q(x)$

$q(x)$  is *necessary* and *sufficient* for  $p(x)$

## Boolean Formulas Quantified

given  $A(x, y)$  is a Boolean formula with  $x$  and  $y$  some propositions, then:

$A$  is a tautology means  $\forall x, y A(x, y)$  (69)

$A$  is a contradiction means  $\forall x, y \neg A(x, y)$  (70)

$A$  is a contradiction means  $\exists x, y A(x, y)$  (71)

here, the universe is  $\{T, F\}$

## Logic with Predicates

All *equivalences* and *implications* work for predicates

## Python Predicates and Quantifiers

```
>>> def p(x,y):          # Any function that returns T/F to be used as predicate
    return x >= y
>>> p(2,1)
# True

>>> def p(x): return x >= 0
>>> S = { -1, 0, 1 }; T = { 0, 1, 2 }
>>> Sp = [ p(x) for x in S ] # List of booleans
>>> Tp = [ p(x) for x in T ] # List of booleans
>>> all(Tp)                  # Check for all: ALL True
# True
>>> all(Sp)                  # Check there exists: at least ONE True
# True
>>> any(p(x) for x in S)     # Generator expression
# True
```