

# Discrete Math for Computer Science

Peter Schaefer

Freshman Fall

## Contents

<b>1</b>	<b>Logic</b>	<b>4</b>
1.1	Propositions and Logical Operations . . . . .	4
1.2	Evaluating Compound Propositions . . . . .	4
1.3	Conditional Statements . . . . .	4
1.4	Logical Equivalence . . . . .	4
1.5	Laws of Propositional Logic . . . . .	5
1.6	Predicates and Quantifiers . . . . .	5
1.7	Quantified Statements . . . . .	5
1.8	DeMorgan's law for Quantified Statements . . . . .	6
1.9	Nested Quantifiers . . . . .	6
1.10	More Nested Quantifiers . . . . .	6
1.11	Logical Reasoning . . . . .	7
1.12	Rules of Inference with Propositions . . . . .	8
1.13	Rules of Inference with Quantifiers . . . . .	8
<b>2</b>	<b>Proofs</b>	<b>10</b>
2.1	Mathematical Definitions . . . . .	10
2.2	Introduction to Proofs . . . . .	10
2.3	Writing Proofs: Best Practices . . . . .	10
2.4	Writing Direct Proofs . . . . .	10
2.5	Proof by Contrapositive . . . . .	10
2.6	Proof by Contradiction . . . . .	10
2.7	Proof by Cases . . . . .	10
<b>3</b>	<b>Sets</b>	<b>11</b>
3.1	Sets and Subsets . . . . .	11
3.2	Sets of sets . . . . .	12
3.3	Union and Intersection . . . . .	12
3.4	More set operations . . . . .	13
3.5	Set identities . . . . .	14
3.6	Cartesian products . . . . .	14
3.7	Partitions . . . . .	15
<b>4</b>	<b>Functions</b>	<b>16</b>
4.1	Definition of functions . . . . .	16
4.2	Floor and Ceiling functions . . . . .	16
4.3	Properties of functions . . . . .	17
4.4	The inverse of a function . . . . .	17
4.5	Composition of functions . . . . .	18
4.5.1	Identity Function . . . . .	18
4.6	Logarithms and exponents . . . . .	18

<b>5</b>	<b>Boolean Algebra</b>	<b>20</b>
5.1	An introduction to Boolean Algebra . . . . .	20
5.2	Boolean functions . . . . .	21
5.3	Disjunctive and conjunctive normal form . . . . .	21
5.4	Functional completeness . . . . .	22
5.5	Boolean satisfiability . . . . .	22
5.6	Gates and circuits . . . . .	22
<b>6</b>	<b>Relation and Digraphs</b>	<b>25</b>
6.1	Introduction to binary relations . . . . .	25
6.2	Properties of binary relations . . . . .	26
6.3	Directed graphs, paths, and cycles . . . . .	27
6.4	Composition of relations . . . . .	29
6.5	Graph powers and the transitive closure . . . . .	29
6.6	Matrix multiplication and graph powers . . . . .	30
6.7	Partial orders . . . . .	32
6.8	Strict orders and directed acyclic graphs . . . . .	34
6.9	Equivalence relations . . . . .	34
6.10	N-ary relations and relational databases . . . . .	34
<b>7</b>	<b>Computation</b>	<b>35</b>
7.1	An introduction to algorithms . . . . .	35
7.2	Asymptotic growth of functions . . . . .	35
7.3	Analysis of algorithms . . . . .	35
7.4	Finite state machines . . . . .	35
7.5	Turing machines . . . . .	35
7.6	Decision problems and languages . . . . .	35
<b>8</b>	<b>Induction and Recursion</b>	<b>36</b>
8.1	Sequences . . . . .	36
8.2	Recurrence relations . . . . .	36
8.3	Summations . . . . .	36
8.4	Mathematical induction . . . . .	36
8.5	More inductive proofs . . . . .	36
8.6	Strong induction and well-ordering . . . . .	36
8.7	Loop invariants . . . . .	36
8.8	Recursive definitions . . . . .	36
8.9	Structural induction . . . . .	36
8.10	Recursive algorithms . . . . .	36
8.11	Induction and recursive algorithms . . . . .	36
8.12	Analyzing the time complexity of recursive algorithms . . . . .	36
8.13	Divide-and-conquer algorithms: Introduction and mergesort . . . . .	36
8.14	Divide-and-conquer algorithms: Binary Search . . . . .	36
8.15	Solving linear homogeneous recurrence relations . . . . .	36
8.16	Solving linear non-homogeneous recurrence relations . . . . .	36
8.17	Divide-and-conquer recurrence relations . . . . .	36
<b>9</b>	<b>Integer Properties</b>	<b>37</b>
9.1	The Division Algorithm . . . . .	37
9.2	Modular arithmetic . . . . .	37
9.3	Prime factorizations . . . . .	37
9.4	Factoring and primality testing . . . . .	37
9.5	Greatest common factor divisor and Euclid's algorithm . . . . .	37
9.6	Number representation . . . . .	37
9.7	Fast exponentiation . . . . .	37

9.8	Introduction to cryptography . . . . .	37
9.9	The RSA cryptosystem . . . . .	37
<b>10</b>	<b>Introduction to Counting</b>	<b>38</b>
10.1	Sum and Product Rules . . . . .	38
10.2	The Bijection Rules . . . . .	38
10.3	The generalized product rule . . . . .	38
10.4	Counting permutations . . . . .	38
10.5	Counting subsets . . . . .	38
10.6	Subset and permutation examples . . . . .	38
10.7	Counting by complement . . . . .	38
10.8	Permutations with repetitions . . . . .	38
10.9	Counting multisets . . . . .	38
10.10	Assignment problems: Balls in bins . . . . .	38
10.11	Inclusion-exclusion principle . . . . .	38
<b>11</b>	<b>Advanced Counting</b>	<b>39</b>
11.1	Generating permutations . . . . .	39
11.2	Binomial coefficients and combinatorial identities . . . . .	39
11.3	The pigeonhole principle . . . . .	39
11.4	Generating functions . . . . .	39
<b>12</b>	<b>Discrete Probability</b>	<b>40</b>
12.1	Probability of an event . . . . .	40
12.2	Unions and complements of events . . . . .	40
12.3	Conditional probability and independence . . . . .	40
12.4	Bayes' Theorem . . . . .	40
12.5	Random variables . . . . .	40
12.6	Expectation of random variables . . . . .	40
12.7	Linearity of expectations . . . . .	40
12.8	Bernoulli trials and the binomial distribution . . . . .	40
<b>13</b>	<b>Graphs</b>	<b>41</b>
13.1	Introduction to Graphs . . . . .	41
13.2	Graph representations . . . . .	41
13.3	Graph isomorphism . . . . .	41
13.4	Walks, trails, circuits, paths, and cycles . . . . .	41
13.5	Graph connectivity . . . . .	41
13.6	Euler circuits and trails . . . . .	41
13.7	Hamiltonian cycles and paths . . . . .	41
13.8	Planar coloring . . . . .	41
13.9	Graph coloring . . . . .	41
<b>14</b>	<b>Trees</b>	<b>42</b>
14.1	Introduction to trees . . . . .	42
14.2	Tree application examples . . . . .	42
14.3	Properties of trees . . . . .	42
14.4	Tree traversals . . . . .	42
14.5	Spanning trees and graph traversals . . . . .	42
14.6	Minimum spanning trees . . . . .	42

# 1 Logic

## 1.1 Propositions and Logical Operations

**Proposition:** a statement that is either true or false.

Some examples include "It is raining today" and " $3 \cdot 8 = 20$ ".

However, not all statements are propositions, such as "open the door"

Name	Symbol	alternate name	$p$	$q$	$\neg p$	$p \wedge q$	$p \vee q$	$p \oplus q$
NOT	$\neg$	negation	T	T	F	T	T	F
AND	$\wedge$	conjunction	T	F	F	F	T	T
OR	$\vee$	disjunction	F	T	T	F	T	T
XOR	$\oplus$	exclusive or	F	F	T	F	F	F

XOR is very useful for encryption and binary arithmetic.

## 1.2 Evaluating Compound Propositions

$p$ : The weather is bad.

$p \wedge q$ : The weather is bad *and* the trip is cancelled

$q$ : The trip is cancelled.

$p \vee q$ : The weather is bad *or* the trip is cancelled

$r$ : The trip is delayed.

$p \wedge (q \oplus r)$ : The weather is bad *and* either the trip is cancelled *or* delayed

**Order of Evaluation**  $\neg$ , then  $\wedge$ , then  $\vee$ , but parenthesis always help for clarity.

Example Truth Table:

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

## 1.3 Conditional Statements

$p \implies q$  where  $p$  is the hypothesis and  $q$  is the conclusion

Format	Terminology	
$p \implies q$	given	given
$\neg q \implies \neg p$	contrapositive	$p \implies q \equiv \neg q \implies \neg p$ contrapositive
$q \implies p$	converse	inverse
$\neg p \implies \neg q$	inverse	$\neg p \implies \neg q \equiv q \implies p$ converse

$p$	$q$	$p \implies q$		Phrase	Logic
T	T	T	$p$ is a <u>sufficient</u> condition for $q$	$q$ if $p$	$p \implies q$
T	F	F	$q$ is a <u>necessary</u> condition for $p$	$q$ only if $p$	$q \implies p$
F	T	T		$q$ if and only if $p$	$p \iff q$
F	F	T			

**Order of Operations:**  $p \wedge q \implies r \equiv (p \wedge q) \implies r$

## 1.4 Logical Equivalence

**Tautology:** a proposition that is always true

**Contradiction:** a proposition that is always false

**Logically equivalent:** same truth value regardless of the truth values of their individual propositions

**DeMorgan's Laws:**

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

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

Verbally,

It is not true that the patient has migraines *or* high blood pressure  $\equiv$   
 $\equiv$  The patient does not have migraines *and* does not have high blood pressure  


---

It is not true that the patient has migraines *and* high blood pressure  $\equiv$   
 $\equiv$  The patient does not have migraines *or* does not have high blood pressure

## 1.5 Laws of Propositional Logic

You can use **substitution** on logically equivalent propositions.

Law Name	$\vee$ or	$\wedge$ and
Idempotent	$p \vee p \equiv p$	$p \wedge p \equiv p$
Associative	$(p \vee q) \vee r \equiv p \vee (q \vee r)$	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Commutative	$p \vee q \equiv q \vee p$	$p \wedge q \equiv q \wedge p$
Distributive	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
Identity	$p \vee F \equiv p$	$p \wedge T \equiv p$
Domination	$p \vee T \equiv T$	$p \wedge F \equiv F$
Double Negation	$\neg \neg p \equiv p$	
Complement	$p \vee \neg p \equiv T$	$p \wedge \neg p \equiv F$
DeMorgan	$\neg(p \vee q) \equiv \neg p \wedge \neg q$	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
Absorption	$p \vee (p \wedge q) \equiv p$	$p \wedge (p \vee q) \equiv p$
Conditional	$p \implies q \equiv \neg p \vee q$	$p \iff q \equiv (p \implies q) \wedge (q \implies p)$

## 1.6 Predicates and Quantifiers

**Predicate:** a logical statement where truth value is a function of a variable.

$P(x)$ :  $x$  is an even number.       $P(5)$ : false       $P(2)$ : true

**Domain:** the set of all possible values for a variable in a predicate.

Ex.  $\mathbb{Z}^+$  is the set of all positive integers.

\*If domain is not clear from context, it should be given as part of the definition of the predicate.

Quantifier	Symbol	Meaning
Universal	$\forall$	"for all"
Existential	$\exists$	"there exists"

**Quantifier:** converts a predicate to a proposition.

$\exists x(x + 1 < x)$  is false.

**Counter Example:** universally quantified statement where an element in the domain for which the predicate is false. Useful to prove a  $\forall$  statement false.

## 1.7 Quantified Statements

Consider the two following two predicates:

$P(x)$ :  $x$  is prime,  $x \in \mathbb{Z}^+$

$O(x)$ :  $x$  is odd

Proposition made of predicates:  $\exists x(P(x) \wedge \neg O(x))$

Verbally: there exists a positive integer that is prime but is not odd.

**Free Variable:** a variable that is free to be any value in the domain.

**Bound Variable:** a variable that is bound to a quantifier.

			$P(x)$	$S(x)$	$\neg S(x)$
$P(x)$ :	$x$ came to the party	$P(x) \stackrel{?}{\equiv} \neg S(x)$	Joe	T	F
$S(x)$ :	$x$ was sick	$P(x) \not\equiv \neg S(x)$	Theo	F	T
			Gert	T	F
			Sam	F	T

## 1.8 DeMorgan's law for Quantified Statements

Consider the predicate:  $F(x) : "x \text{ can fly}"$ , where  $x$  is a bird. According to the DeMorgan Identity for Quantified Statements,

$$\neg \forall x F(x) \equiv \exists x \neg F(x)$$

"not every bird can fly  $\equiv$  "there exists a bird that cannot fly"

Example using DeMorgan Identities:

$$\begin{aligned} \neg \exists x (P(x) \implies \neg Q(x)) &\equiv \forall x \neg (P(x) \implies \neg Q(x)) \\ &\equiv \forall x (\neg \neg P(x) \wedge \neg \neg Q(x)) \\ &\equiv \forall x (P(x) \wedge Q(x)) \end{aligned}$$

## 1.9 Nested Quantifiers

A logical expression with more than one quantifier that binds different variables in the same predicate is said to have **Nested Quantifiers**.

Logic	Variable Boundedness	Logic	Meaning
$\forall x \exists y P(x, y)$	$x, y$ bound	$\forall x \forall y M(x, y)$	"everyone sent an email to everyone"
$\forall x P(x, y)$	$x$ bound, $y$ free	$\forall x \exists y M(x, y)$	"everyone sent an email to someone"
$\exists x \exists y T(x, y, z)$	$x, y$ bound, $z$ free	$\exists x \forall y M(x, y)$	"someone sent an email to everyone"
		$\exists x \exists y M(x, y)$	"someone sent an email to someone"

There is a two-player game analogy for how quantifiers work:

Player	Action	Goal
Existential Player $\exists$	selects value for existentially-bound variables	tries to make expression <u>true</u>
Universal Player $\forall$	selects value for universally-bound variables	tries to make expression <u>false</u>

Consider the predicate  $L(x, y) : "x \text{ likes } y"$ .

$\exists x \forall y L(x, y)$  means "there is a student who likes everyone in the school".

$\neg \exists x \forall y L(x, y)$  means "there is no student who likes everyone in the school".

After applying DeMorgan's Laws,

$\forall x \exists y \neg L(x, y)$  means "there is no student who likes everyone in the school".

## 1.10 More Nested Quantifiers

$M(x, y) : "x \text{ sent an email to } y"$ . Consider  $\forall x \forall y M(x, y)$ . It means that "email sent an email to everyone including themselves". Using  $(x \neq y \implies M(x, y))$  can fix this quirk.

$\forall x \forall y (x \neq y \implies M(x, y))$  means "everyone sent an email to everyone else"

### Expressing Uniqueness in Quantified Statements

Consider  $L(x)$ :  $x$  was late to the meeting. If someone was late to the meeting, how could you express that that someone was the only person late to the meeting? You want to express that there is someone where everyone else was not late, which can be done with

$$\exists x (L(x) \wedge \forall y (x \neq y \implies \neg L(y)))$$

### Moving Quantifiers in Logical Statements

Consider  $M(x, y)$ : " $x$  is married to  $y$ " and  $A(x)$ : " $x$  is an adult". One way of expressing "For every person  $x$ , if  $x$  is an adult, then there is a person  $y$  to whom  $x$  is married to" is by this statement:

$$\forall x (A(x) \implies \exists y M(x, y))$$

Since  $y$  does not appear in  $A(x)$ , " $\exists y$ " can be moved so that it appears just after the " $\forall$ ", resulting with

$$\forall x \exists y (A(x) \implies M(x, y))$$

When doing this, keep in mind that  $\forall x \exists y \neq \exists y \forall x$ :

$$\forall x \exists y (A(x) \implies M(x, y)) \text{ means}$$

for every  $x$ , if  $x$  is an adult, there exists  $y$  who is married to  $x$ .

$$\exists y \forall x (A(x) \implies M(x, y)) \text{ means}$$

There exists a  $y$ , such that every  $x$  who is an adult is also married to  $y$

## 1.11 Logical Reasoning

**Argument:** a sequence of propositions, called hypothesis, followed by a final proposition, called the conclusion.

An argument is **valid** if the conclusion is true whenever the hypothesis are all true, otherwise the argument is **invalid**.

$$\begin{array}{c} p_1 \\ p_2 \\ \vdots \\ p_n \\ \hline \therefore c \end{array} \quad \text{where } \begin{array}{l} p_1, p_2, \dots, p_n \text{ are hypothesis} \\ c \text{ is the conclusion} \end{array}$$

The argument is valid whenever the proposition  $(p_1 \wedge p_2 \wedge \dots \wedge p_n) \implies c$  is a tautology. Additionally, because of the commutative law, hypothesis can be reordered without changing the argument.

$$\frac{p}{p \implies q} \quad \equiv \quad \frac{p \implies q}{p} \quad \therefore q$$

### The Form of an Argument

$$\begin{array}{l} \text{It is raining today.} \\ \text{If it is raining today, then I will not ride my bike to school.} \\ \hline \therefore \text{I will not ride my bike to school.} \end{array} \quad \frac{p}{p \implies q} \quad \therefore q$$

$$\text{The argument is valid because its form, } \frac{p}{p \implies q} \text{ is an valid argument.}$$

$$\begin{array}{l} 5 \text{ is not an even number.} \\ \text{If 5 is an even number, then 7 is an even number.} \\ \hline \therefore 7 \text{ is not an even number.} \end{array} \quad \frac{p}{p \implies q} \quad \therefore q$$

$$\text{The argument is invalid because its form, } \frac{\neg p}{p \implies q} \text{ is an invalid argument.}$$

## 1.12 Rules of Inference with Propositions

Using truth tables to establish the validity of an argument can become tedious, especially if an argument uses a large number of variables.

$\frac{p \quad p \implies q}{\therefore q}$	Modus Ponens	$\frac{p \quad q}{\therefore p \wedge q}$	Conjunction
$\frac{\neg q \quad p \implies q}{\therefore \neg p}$	Modus Tollens	$\frac{p \implies q \quad q \implies r}{\therefore p \implies r}$	Hypothetical Syllogism
$\frac{p}{\therefore p \vee q}$	Addition	$\frac{p \vee q \quad \neg p}{\therefore q}$	Disjunctive Syllogism
$\frac{p \wedge q}{\therefore p}$	Simplification	$\frac{p \implies q \quad q \implies r}{\therefore q \vee r}$	Resolution

Example expressed in English:

If it is raining or windy or both, the game will be cancelled.	$(r \vee w) \implies c$
The game will not be cancelled	$\neg c$
$\therefore$ It is not windy.	$\therefore \neg w$

Steps to Solve:

$(r \vee w) \implies c$	Hypothesis	(1)
$\neg c$	Hypothesis	(2)
$\neg(r \vee w)$	Modus Tollens: 1, 2	(3)
$\neg r \wedge \neg w$	DeMorgan's Law: 3	(4)
$\neg w \wedge \neg r$	Commutative Law: 4	(5)
$\neg w$	Simplification: 5	(6)

## 1.13 Rules of Inference with Quantifiers

In order to apply the rules of quantified expressions, such as  $\forall x \neg (P(x) \wedge Q(x))$ , we need to remove the quantifier by plugging in a value from the domain to replace the variable x.

For example:

Every employee who received a large bonus works hard.	$\forall x (B(x) \implies H(x))$
Linda is an employee at the company.	$Linda \in x$
Linda received a large bonus.	$B(Linda)$
$\therefore$ Some employee works hard.	$\therefore \exists x H(x)$

**Arbitrary Element:** has no special properties other than those shared by all elements of the domain.

**Particular Element:** may have special properties that are not shared by all the elements of the domain. For example, if the domain is the set of all integers,  $\mathbb{Z}$ , a particular element is 3, because it is odd, which is not true for all integers.



**Rules of Inference for Quantified Statements**

$c$ is an element $\frac{\forall x P(x)}{\therefore P(c)}$	Universal Instantiation	$\frac{\exists x P(x)}{\therefore c \text{ is particular} \wedge P(c)}$	Existential Instantiation*
$c$ is arbitrary $\frac{P(c)}{\therefore \forall P(x)}$	Universal Generalization	$\frac{c \text{ is an element} \quad P(c)}{\therefore \exists x P(x)}$	Existential Generalization

\*Each use of Existential Instantiation must define a new element with its own symbol or name.

**Example of using the Laws of Inference for Quantified Statements**

Consider the following argument:

$$\frac{\begin{array}{l} \forall x(P(x) \vee Q(x)) \\ 3 \text{ is an integer} \\ \neg P(3) \end{array}}{\therefore Q(3)}$$

Steps to Solve:

$\forall x(P(x) \vee Q(x))$	Hypothesis	(1)
3 is an integer	Hypothesis	(2)
$(P(3) \vee Q(3))$	Universal Instantiation: 1, 2	(3)
$\neg P(3)$	Hypothesis	(4)
$Q(3)$	Disjunctive Syllogism: 3, 4	(5)

**Showing an Argument with Quantified Statements is Invalid**

Consider the following argument:

$$\frac{\begin{array}{l} \exists x P(x) \\ \exists x Q(x) \end{array}}{\therefore \exists x(P(x) \wedge Q(x))}$$

Using a supposed domain  $\{c, d\}$ , with truth values of

	P	Q
c	T	F
d	F	T

, the argument is invalid.

## 2 Proofs

### 2.1 Mathematical Definitions

### 2.2 Introduction to Proofs

### 2.3 Writing Proofs: Best Practices

### 2.4 Writing Direct Proofs

### 2.5 Proof by Contrapositive

### 2.6 Proof by Contradiction

### 2.7 Proof by Cases

## 3 Sets

### 3.1 Sets and Subsets

A **set** is a collection of objects. Objects in a set are called **elements**. Order does not matter, and there are no duplicates.

Roster notation:

$$A = \{2, 4, 6, 10\}$$

$$B = \{4, 6, 10, 2\}$$

$$A = B$$

To show membership, use the  $\in$  symbol. For example,  $2 \in A$ , while  $7 \notin A$ . The empty set, which contains nothing, typically uses the  $\emptyset$  symbol, or  $\{\}$ . Sets can be finite, or infinite. **Cardinality** of a set is the number of elements in a set. For example, the cardinality of  $A$  is 4.

$$|A| = 4$$

Cardinality can be infinite. Consider the set of all the integers,  $\mathbb{Z}$ .  $|\mathbb{Z}| = \infty$

$\mathbb{N}$  : set of natural numbers

$$= \{0, 1, 2, 3, \dots\}$$

$\mathbb{Z}$  : set of integers

$$= \{\dots, -2, -1, 0, 1, 2, \dots\}$$

$\mathbb{B}$  : set of rational numbers

$$= \{x | x = \frac{a}{b} \text{ where } a, b \in \mathbb{Z}, b \neq 0\}$$

$\mathbb{R}$  : set of real numbers

$$= \{x | x \text{ has a decimal representation}\}$$

The subset operator is  $\subseteq$

$$A \subseteq B \text{ if } \forall x(x \in A \implies x \in B)$$

$$A \subseteq A \text{ is true for any set}$$

$$\emptyset \subseteq A \text{ is true for any set}$$

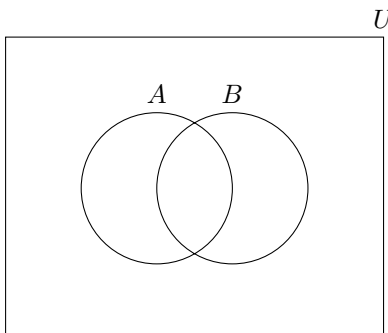
Sometimes it is easier to define a set by defining properties that all the elements have. That is easy to do in **set builder notation**.

$$A = \{x \in S : A(x)\}, \text{ where } S \text{ is another set}$$

$$C = \{x \in \mathbb{Z} : 0 < x < 100 \text{ and } x \text{ is prime}\}.$$

$$D = \{x \in \mathbb{R} : |x| < 1\}$$

The **Universal Set**, usually called 'U', is a set that contains all elements mentioned in a particular context. For example, a discussion about certain types of real numbers, it would be understood that any element in the discussion is a real number. Sets are often represented pictorially with **Venn Diagrams**.



If  $A \subseteq B$  and there is an element of  $B$  that is not an element of  $A$ , meaning  $A \neq B$ , then  $A$  is a **proper subset** of  $B$ , denoted as  $A \subset B$ . An important fact is that  $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{B} \subset \mathbb{R}$

### 3.2 Sets of sets

Elements of sets can be sets themselves, consider  $A = \{\{1, 2\}, \emptyset, \{1, 2, 3\}, \{1\}\}$ . The cardinality of  $A$  is 4,  $|A| = 4$ . Additionally,  $\{1, 2\} \in A$ , but  $1 \notin A$ .

The **Powerset** of  $A$ , denoted as  $P(A)$  is the set of all subsets of  $A$ . For example,

$$A = \{1, 2, 3\}$$

$$P(A) = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

#### Cardinality of a Powerset

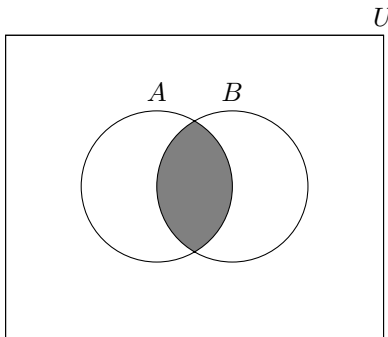
Let  $A$  be a finite set of cardinality  $n$ . Then the cardinality of the powerset of  $A$  is  $2^n$ .

$$|A| = n$$

$$|P(A)| = 2^n$$

### 3.3 Union and Intersection

**Intersection** set operation:  $\cap$ .  $A$  intersected with  $B$  is defined to be the set containing elements which are in both  $A$  and  $B$ . That is,  $A \cap B = \{x : x \in A \wedge x \in B\}$ .



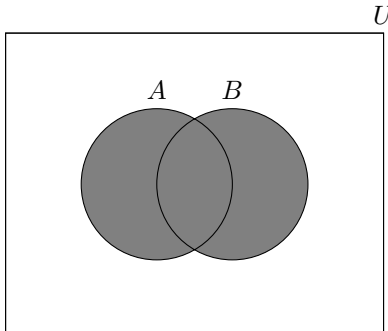
Intersection  $\cap$  can also apply to infinite sets:

$$A = \{x \in \mathbb{Z} : x \text{ is an integer multiple of } 2\}$$

$$B = \{x \in \mathbb{Z} : x \text{ is an integer multiple of } 3\}$$

$$A \cap B = \{x \in \mathbb{Z} : x \text{ is an integer multiple of } 6\}$$

**Union** set operation  $\cup$ .  $A$  union with  $B$  is defined to be the set containing elements which are in  $A$  or  $B$ . That is,  $A \cup B = \{x : x \in A \vee x \in B\}$ .



A special notation, similar to  $\sum$  or  $\prod$  notation, allows for compound representation of the intersections or unions of a long sequence of sets.

$$\bigcap_{i=1}^n A_i = A_1 \cap A_2 \cap A_3 \cap \cdots \cap A_n = \{x : x \in A, \text{ for } \underline{\text{all}} \ 1 \leq i \leq n\}$$

$$\bigcup_{i=1}^n A_i = A_1 \cup A_2 \cup A_3 \cup \cdots \cup A_n = \{x : x \in A, \text{ for } \underline{\text{some}} \ 1 \leq i \leq n\}$$

Consider  $A_j$  = a word with  $j$  letters, with  $U$  = is the Oxford English Dictionary.

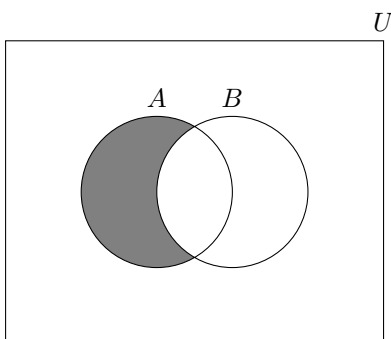
$$\bigcup_{j=1}^{10} A_j = \text{the set of all words with 10 letters or fewer in the OED}$$

$$\bigcap_{j=1}^{45} A_j = \emptyset$$

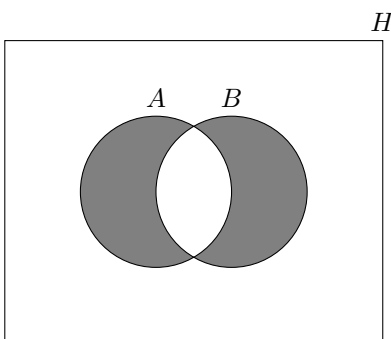
$$\bigcup_{j=1}^{45} A_j = \text{the set of all words in the OED.}$$

### 3.4 More set operations

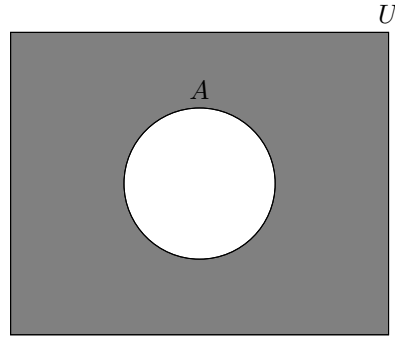
**Difference** set operation  $-$ .  $A$  difference with  $B$  is defined to be the set containing elements which are in  $A$  but not  $B$ . That is,  $A - B = \{x : x \in A \wedge x \notin B\}$ . A set difference is not strictly commutative, often  $A - B \neq B - A$ .



**Symmetric Difference** set operation  $\Delta$ . A symmetric difference with  $B$  is defined to be the set containing elements which are in  $A$  or  $B$ , but not  $A$  and  $B$ . That is,  $A \Delta B = \{x : x \in A \oplus x \in B\}$ .



**Complement** set operation  $\bar{\phantom{x}}$ . complement  $A$  is defined to be the set containing elements in  $U$  which are not in  $A$ . That is,  $\bar{A} = \{x : x \in U \wedge x \notin A\}$ .



Summary of Set Operations

Operation	Notation	Set Builder
Intersection	$A \cap B$	$\{x : x \in A \wedge x \in B\}$
Union	$A \cup B$	$\{x : x \in A \vee x \in B\}$
Difference	$A - B$	$\{x : x \in A \wedge x \notin B\}$
Symmetric Difference	$A \triangle B$	$\{x : x \in A \oplus x \in B\}$
Complement	$\overline{A}$	$\{x : x \in U \wedge x \notin A\}$

### 3.5 Set identities

The laws of propositional logic can be used to derive corresponding set identities. A **set identity** is an equation involving sets that is true, regardless of the contents of the sets used in the expression.

Law Name	$\cup$ Union	$\cap$ Intersection
Idempotent	$A \cup A = A$	$A \cap A = A$
Associative	$(A \cup B) \cup C = A \cup (B \cup C)$	$(A \cap B) \cap C = A \cap (B \cap C)$
Commutative	$A \cup B = B \cup A$	$A \cap B = B \cap A$
Distributive	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
Identity	$A \cup \emptyset = A$	$A \cap U = A$
Domination	$A \cup U = U$	$A \cap \emptyset = \emptyset$
Double Complement	$\overline{\overline{A}} = A$	
Complement	$A \cup \overline{A} = U$	$A \cap \overline{A} = \emptyset$
DeMorgan	$\overline{A \cup B} = \overline{A} \cap \overline{B}$	$\overline{A \cap B} = \overline{A} \cup \overline{B}$
Absorption	$A \cup (A \cap B) = A$	$A \cap (A \cup B) = A$

### 3.6 Cartesian products

An **ordered pair** of items is written  $(x, y)$ , where the first entry is  $x$  and the second entry is  $y$ . The use of  $()$  instead of  $\{\}$  indicates that order matters.

**Cartesian Product** of  $A$  and  $B$ ,  $A \times B = \{(a, b) : a \in A \wedge b \in B\}$

$$\begin{aligned}
 A &= \{1, 2\} & A \times B &= \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c)\} \\
 B &= \{a, b, c\} & B \times A &= \{(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}
 \end{aligned}$$

An ordered list of 3 items is called an **ordered triple**, denoted as  $(x, y, z)$ . For a size of  $\geq 4$ , use the term **n-tuple**. For example,  $(u, w, x, y, z)$ .

$$A_1 \times A_2 \times \cdots \times A_n = \{(a_1, a_2, \dots, a_n) : a_i \in A \text{ for all } i \text{ such that } 1 \leq i \leq n\}$$

Another Example

$$\begin{aligned}
 A &= \{a, b\} & (a, 1, y, \beta) &\in A \times B \times C \times D \\
 B &= \{1, 2\} & (b, 1, x, \alpha) &\in A \times B \times C \times D \\
 C &= \{x, y\} & (1, b, x, \beta) &\notin A \times B \times C \times D \\
 D &= \{\alpha, \beta\} & &\text{order matters}
 \end{aligned}$$

$A \times A = A^2$ , and in general,

$$A^k = \underbrace{A \times A \times \cdots \times A}_{k\text{-times}}$$

The **Cardinality of Cartesian Products**:

$$|A^n| = |A|^n$$

$$|A_1 \times A_2 \times \cdots| = |A_1| \cdot |A_2| \cdots$$

### Strings

A sequence of characters is called a **string**. The set of characters used in a set of string is called the **alphabet** for the set of strings. The **length** of a string is the number of characters in the string. For example, the length of 'xyxyx' is 6. The **empty string** is a string whose length is 0, and is usually denoted by  $\lambda$ . It is useful for  $A^0$ , for some alphabet  $A$ .  $\{0, 1\}^0 = \{\lambda\}$ . If  $s$  and  $t$  are two strings, then the **concatenation** of  $s$  and  $t$  is the string obtained by putting  $s$  and  $t$  together.

$$s = 010$$

$$t = 11$$

$$st = 01011$$

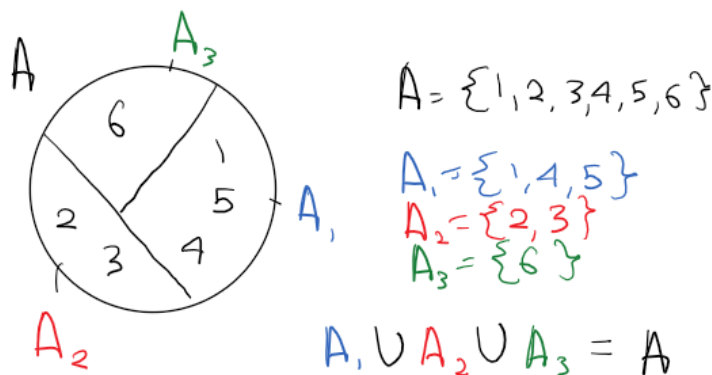
$$t0 = 110$$

Strings are used to specify passwords for computers or online accounts. Security systems vary with respect to the alphabet of characters allowed or required in a valid password. Strings also play an important rules in discrete mathematics as a mathematical tool to help count cardinality of sets.

### 3.7 Partitions

Two sets,  $A$  and  $B$ , are said to be **disjoint** if their intersection is empty ( $A \cap B = \emptyset$ ). A sequence of sets,  $A_1, A_2, A_3, \dots, A_n$ , is **pairwise disjoint** if every pair of distinct sets in the sequence is disjoint. A **partition** of a non-empty set  $A$  is a collection of non-empty subsets such that each element of  $A$  is in exactly one of the subsets.  $A_1, A_2, A_3, \dots, A_n$  is a partition for a nonempty set  $A$  if:

- For all  $i$ ,  $A_i \subseteq A$
- For all  $i$ ,  $A_i \neq \emptyset$
- $A_1, A_2, \dots, A_n$  are pairwise disjoint
- $A = \bigcup_{i=1}^n A_i$ , for some  $n \in \mathbb{Z}^+$



## 4 Functions

### 4.1 Definition of functions

A **function** maps elements of one set  $X$  to elements of another set  $Y$ . A function from  $X$  to  $Y$  can be viewed as a subset of  $X \times Y : (x, y) \in f$  if  $f$  maps  $x$  to  $y$ . The notation for a function is:

$$f : X \rightarrow Y, \text{ where } X \text{ is the } \mathbf{domain} \text{ and } Y \text{ is the } \mathbf{co-domain}.$$

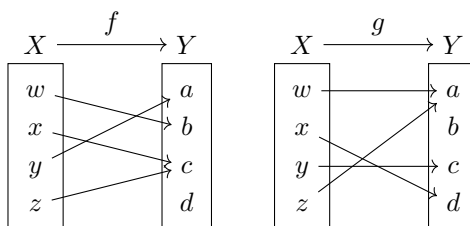
\*if  $f$  maps an element of the domain to zero elements or more than one element of the target, then  $f$  is not *well-defined*

**Arrow Diagram:**

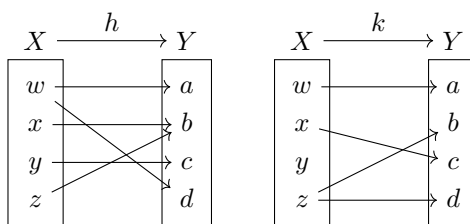
$$X = \{w, x, y, z\}$$

$$Y = \{a, b, c, d\}$$

Well-defined functions:



Not well-defined functions:



For function  $f : X \rightarrow Y$ , an element  $y$  is in the **range** of  $f$  iff there is an  $x \in X$  such that  $(x, y) \in f$ .

$$\text{Range of } f = \{y : (x, y) \in f, \text{ for some } x \in X\}$$

Two functions,  $f$  and  $g$ , are **equal** if  $f$  and  $g$  have the same domain and target and  $f(x) = g(x)$  for every  $x$  in the domain.

$$\forall x : f(x) = g(x) \implies f = g$$

### 4.2 Floor and Ceiling functions

The **Floor** function,  $\lfloor x \rfloor$

$$\text{floor} : \mathbb{R} \rightarrow \mathbb{Z}, \text{ where } \text{floor}(x) = \text{the largest integer } y \text{ such that } y \leq x.$$

Notation:  $\text{floor}(x) = \lfloor x \rfloor$

The **Ceiling** function,  $\lceil x \rceil$

$$\text{ceiling} : \mathbb{R} \rightarrow \mathbb{Z}, \text{ where } \text{ceiling}(x) = \text{the smallest integer } y \text{ such that } y \geq x.$$

Notation:  $\text{ceiling}(x) = \lceil x \rceil$



Examples of floor and ceiling:

$$\begin{array}{ll} \lceil 4.32 \rceil = 5 & \lfloor 4.32 \rfloor = 4 \\ \lceil -4.32 \rceil = -4 & \lfloor -4.32 \rfloor = -5 \\ \lceil 4 \rceil = 4 & \lfloor 4 \rfloor = 4 \\ \lceil -4 \rceil = -4 & \lfloor -4 \rfloor = -4 \end{array}$$

### 4.3 Properties of functions

A function  $f : X \rightarrow Y$  is **one-to-one** or **injective** if  $x_1 \neq x_2$  implies that  $f(x_1) \neq f(x_2)$ .  $f$  maps different elements in  $x$  to different elements in  $y$ .

A function  $f : X \rightarrow Y$  is **onto** or **surjective** if the range of  $f$  is equal to the target  $Y$ . That is,  $\forall y \exists x (y \in Y \wedge x \in X \wedge f(x) = y)$

A function  $f : X \rightarrow Y$  is **bijective** if it is both **injective** and **surjective**. A **bijective** function is called a **bijection**, or a **one-to-one correspondence**.

When the domain and target are finite sets, it is possible to infer information about their relative sizes based on whether a function is one-to-one or onto.

$$\begin{array}{lll} f : D \rightarrow T \text{ is } \mathbf{one-to-one} & \implies & |D| \leq |T| \\ f : D \rightarrow T \text{ is } \mathbf{onto} & \implies & |D| \geq |T| \\ f : D \rightarrow T \text{ is } \mathbf{bijective} & \implies & |D| = |T| \end{array}$$

### 4.4 The inverse of a function

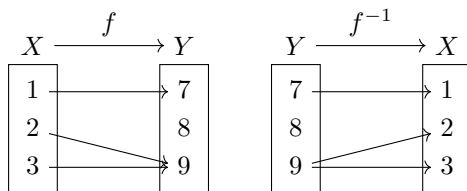
If a function  $f : X \rightarrow Y$  is a *bijection*, then the **inverse** of  $f$  is obtained by exchanging the first and second entries in each pair in  $f$ .

$$\begin{array}{l} \text{given } f : X \rightarrow Y \\ \text{inverse } f^{-1} : \{(y, x) : (x, y) \in f\} \end{array}$$

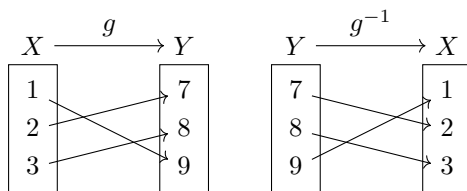
Reversing the cartesian pair does not always create a well-defined function. *Some functions do not have an inverse.*

**Examples:**

$$\begin{array}{ll} X = \{1, 2, 3\} & f = \{(1, 7), (2, 9), (3, 9)\} \\ Y = \{7, 8, 9\} & g = \{(1, 9), (2, 7), (3, 8)\} \end{array}$$



$f^{-1}$  is not well defined, therefore  $f$  does not have an inverse.



$g^{-1}$  is well defined, therefore  $g$  does have an inverse.

## 4.5 Composition of functions

The process of applying a function to the result of another function is called **composition**.

$$\begin{aligned} f &: X \rightarrow Y \\ g &: Y \rightarrow Z \\ (g \circ f) &: X \rightarrow Z, \text{ such that } \forall x : x \in X, (g \circ f)(x) = g(f(x)) \end{aligned}$$

Remember that order matters, as often  $(g \circ f)(x) \neq (f \circ g)(x)$ . However, composition is associative:

$$(f \circ g \circ h)(x) = ((f \circ g) \circ h)(x) = (f \circ (g \circ h))(x) = f(g(h(x)))$$

### 4.5.1 Identity Function

The **Identity Function** maps a set onto itself and maps every element to itself. It is notated as  $I_A : A \rightarrow A$ , where  $A$  is the set it maps. There are a number of identities about the Identity Function.

Let  $f : A \rightarrow B$  be a bijection. Then,

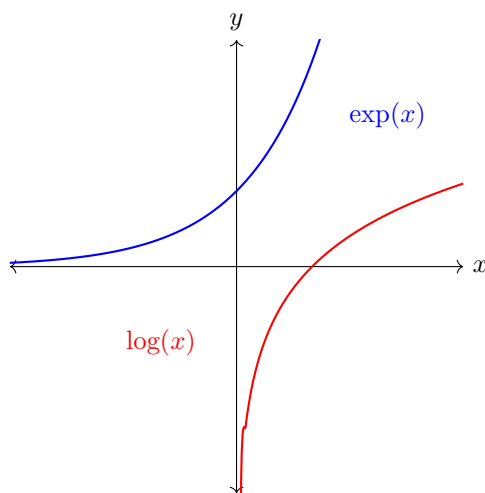
$$f \circ f^{-1} = I_B \text{ and } f^{-1} \circ f = I_A$$

## 4.6 Logarithms and exponents

The **Exponential** function,  $\exp_b : \mathbb{R} \rightarrow \mathbb{R}^+, \exp_b(x) = b^x$ .  $b$  is the base of the exponent and  $x$  is the exponent.

Properties of exponents:

$$\begin{array}{lll} b^x b^y = b^{x+y} & b \in \mathbb{R}^+ & c \in \mathbb{R}^+ \\ (b^x)^y = b^{xy} & x \in \mathbb{R} & y \in \mathbb{R} \\ \frac{b^x}{b^y} = b^{x-y} & & \\ (bc)^x = b^x c^x & & \end{array}$$



The **Logarithms** function,  $\log_b : \mathbb{R} \rightarrow \mathbb{R}^+, \log_b(y) = x$ .  $b$  is the base of the logarithm and  $x$  is the exponent.

Properties of exponents:

$$\log_b(xy) = \log_b x + \log_b y \quad b \in \mathbb{R}^+$$

$$\log_b\left(\frac{x}{y}\right) = \log_b x - \log_b y \quad c \in \mathbb{R}^+$$

$$\log_b(x^y) = y \log_b x \quad x \in \mathbb{R}$$

$$\log_c x = \frac{\log_b x}{\log_b c} \quad y \in \mathbb{R}$$

## 5 Boolean Algebra

### 5.1 An introduction to Boolean Algebra

**Boolean Algebra** is a set of rules/operations for working with variables whose values are either 0 or 1. It corresponds highly to propositional logic.

**Boolean Multiplication**, denoted by  $\cdot$ .

Boolean  $\cdot$

$$0 \cdot 0 = 0$$

$$0 \cdot 1 = 0$$

$$1 \cdot 0 = 0$$

$$1 \cdot 1 = 1$$

Logic  $\wedge$

$$F \wedge F = F$$

$$F \wedge T = F$$

$$T \wedge F = F$$

$$T \wedge T = T$$

**Boolean Addition**, denoted by  $+$ .

Boolean  $+$

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 1$$

Logic  $\vee$

$$F \vee F = F$$

$$F \vee T = T$$

$$T \vee F = T$$

$$T \vee T = T$$

**Boolean Complement**, denoted by  $\bar{\phantom{x}}$ .

Boolean  $\bar{\phantom{x}}$

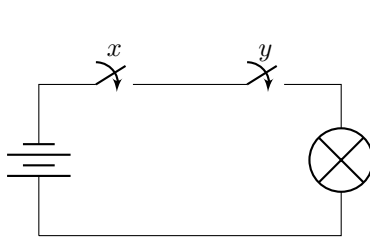
$$\bar{0} = 1$$

$$\bar{1} = 0$$

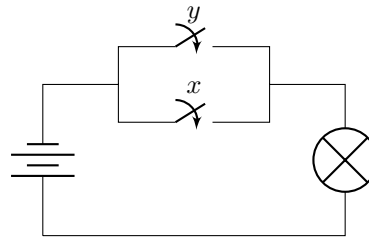
Logic  $\neg$

$$\neg F = T$$

$$\neg T = F$$



Shannon Circuit (AND  $\cdot$ )



Switching Circuit (OR  $+$ )

Variables that can have a value of either 1 or 0 are called **Boolean Variables**. Boolean expressions are made of boolean variables. There are also common shorthand ways of notating operations.

$$x \cdot y + 1 \cdot \bar{z} = xy + \bar{z}$$

$$x + z + \overline{0 + y} = x + z \cdot \bar{y}$$

Law Name	+ OR	· AND
Idempotent	$x + x = x$	$x \cdot x = x$
Associative	$(x + y) + z = x + (y + z)$	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$
Commutative	$x + y = y + x$	$x \cdot y = y \cdot x$
Distributive	$x + (y \cdot z) = (x + y) \cdot (x + z)$	$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
Identity	$x + 0 = x$	$x \cdot 1 = x$
Domination	$x + 1 = 1$	$x \cdot 0 = 0$
Double Complement	$\overline{\overline{x}} = x$	
Complement	$x + \overline{x} = 1$	$x \cdot \overline{x} = 0$
DeMorgan	$\overline{x + y} = \overline{x} \cdot \overline{y}$	$\overline{x \cdot y} = \overline{x} + \overline{y}$
Absorption	$x + (x \cdot y) = x$	$x \cdot (x + y) = x$

## 5.2 Boolean functions

A **boolean function** is a function which maps  $B^k \rightarrow B$ , where  $B = \{0, 1\}$ . For example, consider  $f : B^3 \rightarrow B$

$x$	$y$	$z$	$f(x, y, z)$	$x$	$y$	$z$	$f(x, y, z)$	
0	0	0	0	0	0	0	0	
0	0	1	0	0	0	1	0	
0	1	0	0	0	1	0	0	
0	1	1	1	0	1	1	1	$\bar{x}yz$
1	0	0	1	1	0	0	1	$x\bar{y}\bar{z}$
1	0	1	1	1	0	1	1	$x\bar{y}z$
1	1	0	0	1	1	0	0	
1	1	1	1	1	1	1	1	$xyz$

This function is equivalent to  $f(x, y, z) = x\bar{y} + yz$ . This can be determined using the boolean table and a strategy of combining the cases.

$$\begin{aligned}
 f(x, y, z) &= \bar{x}yz + x\bar{y}\bar{z} + x\bar{y}z + xyz \\
 &= y(\bar{x}z + x\bar{z}) + \bar{y}(x\bar{z} + xz) \\
 &= y(z \cdot 1) + \bar{y}(x \cdot 1) \\
 &= yz + \bar{y}x = \bar{y}x + yz \\
 &= x\bar{y} + yz
 \end{aligned}$$

A **literal** is a boolean variable or the complement of a boolean variable, for example  $x$  or  $\bar{x}$ . In a boolean function whose input variables are  $v_1, v_2, \dots, v_k$ , a *mini-term* is a product of literals  $u_1, u_2, \dots, u_k$ , such that  $u_j$  is either  $v_j$  or  $\bar{v}_j$ .

## 5.3 Disjunctive and conjunctive normal form

A boolean expression that is the sum of literals is said to be in *disjunctive normal form*, **DNF**. It has the following form:

$$c_1 + c_2 + \dots + c_m, \text{ where } c_j \text{ for } j \in \{1, \dots, m\} \text{ is a product of literals.}$$

For example,  $\bar{x}y\bar{z} + xy + w + y\bar{z}w$ . The complement only applies to a single variable and no addition within a term.

A boolean expression that is the product of sums of literals is said to be in *conjunctive normal form*, **CNF**. It has the following form:

$$d_1 + d_2 + \dots + d_m, \text{ where } d_j \text{ for } j \in \{1, \dots, m\} \text{ is a sum of literals.}$$

Each  $d_j$  is called a clause, and complements are only applied to a single variable. Additionally, there is no multiplication within variables. An example is  $(\bar{x} + y + z)(x + \bar{y})(w)(y + \bar{z} + w)$ .

## 5.4 Functional completeness

A set of operators is functionally complete if any boolean function can be expressed using only operations from the set. Two expressions can be added using only multiplication and complement.

$$x + y = \overline{\overline{x}\overline{y}} \text{ DeMorgan's Law}$$

DeMorgan's Law can be extended to more than two boolean variables:

$$x + y + w + z = \overline{\overline{x}\overline{y}\overline{w}\overline{z}}$$

The same can be said about the addition variant of the law:

$$xy = \overline{\overline{x} + \overline{y}}$$

$$xyz = \overline{\overline{x} + \overline{y} + \overline{z}}$$

The **NAND** operation,  $\uparrow$ , and the **NOR** operation,  $\downarrow$ .

$x$	$y$	$x \uparrow y$	$x \downarrow y$
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0

{NAND} is functionally complete, it can create the complement, from which all other possible gates can be created.

$x$	$x \uparrow x$
0	1
1	0

From the complement, AND can be created, and {AND, COMPLEMENT} has already been proven to be functionally complete.

$$xy = \overline{x \uparrow y} = (x \uparrow y) \uparrow (x \uparrow y)$$

## 5.5 Boolean satisfiability

The **Boolean Satisfiability problem**, called the *SAT* for short, takes the boolean expression as an input and asks whether it is possible to set the values of the variables so that the expression evaluates to 1.

If  $\exists x \exists y \dots B(x, y, \dots)$ , then the expression is **satisfiable**

If  $\forall x \forall y \dots \neg B(x, y, \dots)$ , then the expression is **unsatisfiable**

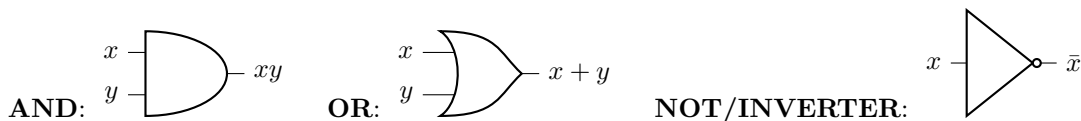
Expressions in DNF form are very easy to determine satisfiability. If there is any term which does *not* contain a variable and its complement, it is satisfiable. For example,

$$x\overline{y}z\overline{x} + \overbrace{\overline{w}xyz}^{\text{no self-complements}} + \overline{w}xw\overline{x} + xy\overline{z}z$$

The above equation *is* satisfiable because there is a term which does not contain a self-complement.

## 5.6 Gates and circuits

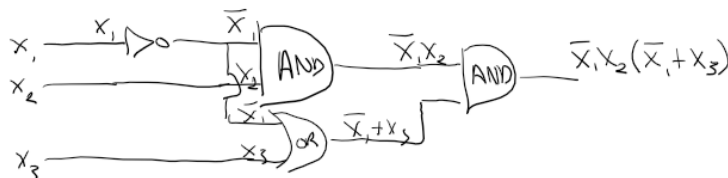
Circuits are built from electrical devices called **gates**.



The boolean function  $f(x_1, x_2) : (f(x_1, x_2) \cdot x_1) + x_2$ . Yes, circuits can contain recursion.

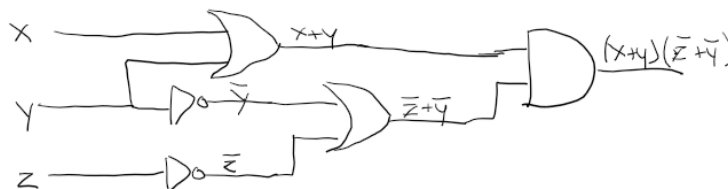


Boolean expressions can also be constructed by following the logic or a circuit.



$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 (\bar{x}_1 + x_3)$$

An example of constructing a circuit from a boolean expression,  $f(x, y, z) : (x + y)(\bar{z} + \bar{y})$



### Designing Circuits

1. Build an input/output table with the desired output for every combination of input
2. Construct a boolean expression that computes the same function as the function specified in the input/output table
3. Construct a digital circuit that realizes the boolean expression

I/O for sum of two bits,  $x, y$

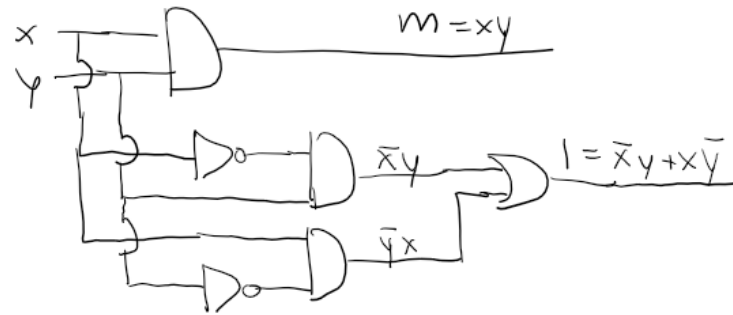
x	y	m	l
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

$$m = xy$$

most significant bit

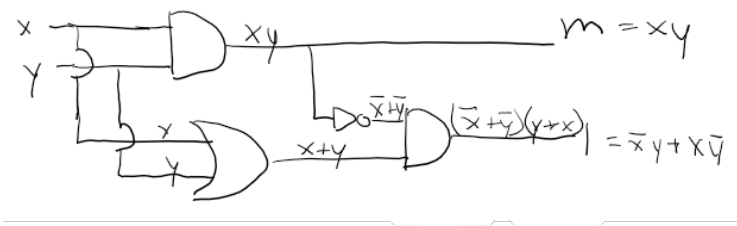
$$l = x\bar{y} + \bar{x}y$$

least significant bit

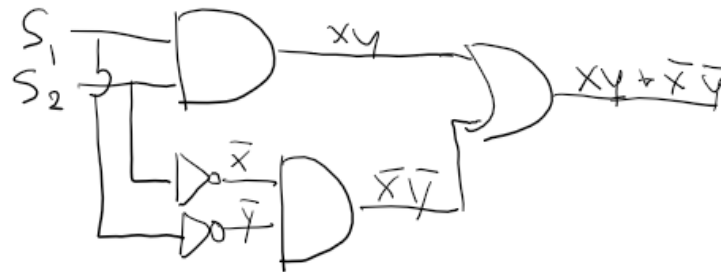


*\*this method does not necessarily give the most efficient circuit.*

Circuits with fewer gates cost less to manufacture. Here is a simplified circuit to add two bits:



Here is another example with boolean logic for light switches:





## 6 Relation and Digraphs

### 6.1 Introduction to binary relations

A **Binary Relation** between two sets  $A$  and  $B$  is a subset  $R$  of  $A \times B$ . It is binary because it is between two sets.

for  $a \in A \wedge b \in B, (a, b) \in R$  is denoted as  $aRb$

For example, consider the relation  $C$  between  $\mathbb{R}$  and  $\mathbb{Z}$ :

$xCy$  if  $|x - y| \leq 1$ , where  $x \in \mathbb{R}$  and  $y \in \mathbb{Z}$

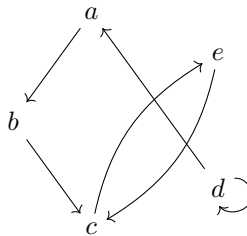
If  $A$  and  $B$  are finite, then relation  $R$  between  $A$  and  $B$  can be represented by a set of ordered pairs.

#### Matrix Representation

$$\begin{aligned}
 P &= \{\text{Sue, Harry, Sam}\} \\
 \text{File} &= \{\text{File A, File B, File C, File D}\} \\
 &\begin{array}{c} \text{File A} \quad \text{File B} \quad \text{File C} \quad \text{File D} \\ \text{Sue} \left( \begin{array}{cccc} 0 & 1 & 1 & 1 \end{array} \right) \\ \text{Harry} \left( \begin{array}{cccc} 1 & 0 & 0 & 0 \end{array} \right) \\ \text{Sam} \left( \begin{array}{cccc} 0 & 0 & 0 & 0 \end{array} \right) \end{array} \\
 \text{An element is} &\quad \begin{array}{l} 1 \text{ if } pRf \text{ is true} \\ 0 \text{ if } pRf \text{ is false} \end{array}
 \end{aligned}$$

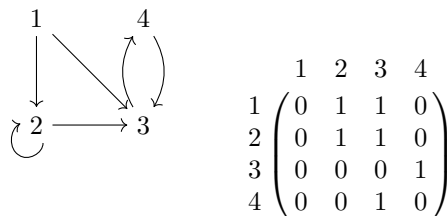
#### Arrow Diagram

$$\begin{aligned}
 A &= \{a, b, c, d, e\} \\
 R &\subseteq A \times A \\
 R &= \{(a, b), (b, c), (e, c), (c, e), (d, a), (d, d)\}
 \end{aligned}$$



#### Arrow Diagram vs. Matrix Representation

$$\begin{aligned}
 A &= \{1, 2, 3, 4\} \\
 R &= \{(1, 2), (1, 3), (2, 2), (2, 3), (3, 4), (4, 3)\}
 \end{aligned}$$



## 6.2 Properties of binary relations

A binary relation of R on set  $A$  is **Reflective** if for *every*  $x \in A$ ,  $xRx$ . For Arrow Diagrams, this means the graph contains self-loops:



For Matrix Representation, this means that the top left to bottom right diagonal are all 1's:

$$\begin{array}{c} a \quad b \quad c \quad d \\ a \begin{pmatrix} 1 & - & - & - \\ - & 1 & - & - \\ - & - & 1 & - \\ - & - & - & 1 \end{pmatrix} \\ b \\ c \\ d \end{array}$$

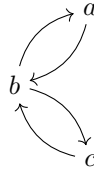
A binary relation of R on set  $A$  is **Anti-reflective** if for *every*  $x \in A$ ,  $xRx$  is *not* true. For Arrow Diagrams, this means the graph does not contain self-loops:



For Matrix Representation, this means that the top left to bottom right diagonal are all 0's:

$$\begin{array}{c} a \quad b \quad c \quad d \\ a \begin{pmatrix} 0 & - & - & - \\ - & 0 & - & - \\ - & - & 0 & - \\ - & - & - & 0 \end{pmatrix} \\ b \\ c \\ d \end{array}$$

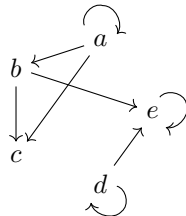
A binary relation of R on set  $A$  is **Symmetric** if and only if for *every* pair  $x \in A$ ,  $y \in Y$ , either *both*  $xRy$  and  $yRx$ , or *both* not  $xRy$  or not  $yRx$  is true. For Arrow Diagrams, this means that every arrow has an arrow going the other way:



For Matrix Representation, this means that the matrix is symmetric along the top left to bottom right diagonal:

$$\begin{array}{c} a \quad b \quad c \quad d \\ a \begin{pmatrix} - & u & v & x \\ u & - & w & y \\ v & w & - & z \\ x & y & z & - \end{pmatrix} \\ b \\ c \\ d \end{array} \quad \text{where} \quad \begin{array}{l} u \in \{0, 1\} \\ \vdots \\ z \in \{0, 1\} \end{array}$$

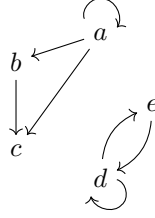
A binary relation of R on set  $A$  is **Anti-symmetric** if and only if for *every* pair  $x \in A$ ,  $y \in Y$ ,  $xRy$  xor  $yRx$ . For Arrow Diagrams, this means that each arrow does not have an arrow going the other way:



For Matrix Representation, this means that the matrix is anti-symmetric along the top left to bottom right diagonal:

$$\begin{matrix} & a & b & c & d \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{pmatrix} - & \bar{u} & \bar{v} & \bar{x} \\ u & - & \bar{w} & \bar{y} \\ v & w & - & \bar{z} \\ x & y & z & - \end{pmatrix} & \text{where} & \begin{matrix} u \in \{0,1\} \\ \vdots \\ z \in \{0,1\} \end{matrix} \end{matrix}$$

A binary relation of  $R$  on set  $A$  is **Transitive** if for *every* three elements  $x, y, z \in A$ , if  $xRy$  and  $yRz$ , then  $xRz$ . Logically,  $(xRy \wedge yRz) \implies xRz$ . For Arrow Diagrams, this means the graph follows a hierarchy or kind of flow:



For Matrix Representation, it is much more difficult to determine transitivity, but here is an example:

$$\begin{matrix} & a & b & c & d & e \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{matrix}$$

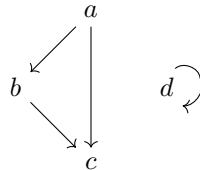
### 6.3 Directed graphs, paths, and cycles

A directed graph, or **Diagram**, consists of a pair  $(V, E)$ , where  $V$  is the set of vertices and  $E$  is the set of *directed edges*. It is a subset of  $V \times V$ .

- indegree: # of edges pointing towards a vertex,  $\text{indegree}(u) = |\{v : (v, u) \in E\}|$
- outdegree: # of edges pointing away from a vertex,  $\text{outdegree}(u) = |\{u : (v, u) \in E\}|$

A digraph is organized into a cartesian pair of the set of vertices and edge pairs:

$$\begin{aligned} \text{Graph } G &= (V, E) \\ V &= \{a, b, c, d\} \\ E &= \{(a, b), (b, c), (a, c), (d, d)\} \end{aligned}$$



$$\text{indegree}(c) = 2$$

$$\text{indegree}(d) = 1$$

$$\text{outdegree}(a) = 2$$

$$\text{outdegree}(d) = 1$$

$a$  is the tail of edge  $(a, b)$

$b$  is the head of edge  $(a, b)$

A digraph is mathematically the same as a relation. Here is an example of the internet as a graph:

Graph  $G = (V, E)$

$V =$  set of all URLs

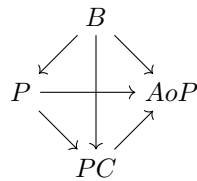
$E =$  set of all hyperlinks from one URL to another URL

$B =$  Blog

$P =$  Pediatrician website

$PC =$  Pharmaceutical Company

$AoP =$  Academy of Pediatrics



### Walks in Directed Graphs

A *walk* is a sequence of vertices and edges. For example, a walk from  $v_0$  to  $v_l$  is notated as:

$$\langle v_0, (v_0, v_1), v_1, (v_1, v_2), \dots, (v_{l-1}, v_l), v_l \rangle$$

where each edge in the sequence appears after its tail and before its head. A walk can also be a set of vertices:

$$\langle v_0, v_1, \dots, v_l \rangle$$

provided that the edges between the vertices *actually exist*. The **length** of a walk is the number of edges traversed.

- **Open walk:** first and last vertices are *not* the same
- **Closed walk:** first and last vertices *are* the same.

### Trails, Circuits, Paths, Cycles

- **Trail:** *open* walk in which no edge occurs more than once.
- **Circuit:** *closed* walk in which no edge occurs more than once. A circuit is a closed trail.
- **Path:** *trail* where no vertex occurs more than once.
- **Cycle:** *circuit* where no vertex occurs more than once, *except* for the first and last, which are the same.

Here are some examples:

$\langle a, c, d, a \rangle$  is a **cycle**: only the first and last vertices are repeated.  
 $\langle a, c, a, d, a \rangle$  is a **circuit**: vertices are repeated, but not edges  
 $\langle a, b, c, b, d \rangle$  is a **trail**: open, vertices are repeated, but not edges

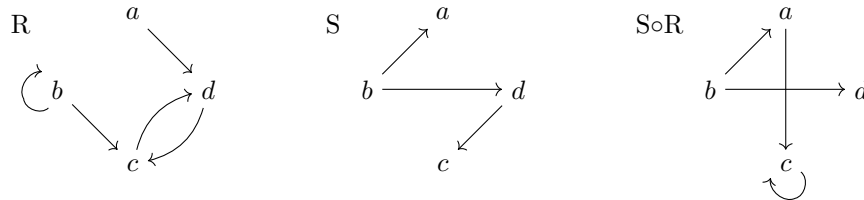
## 6.4 Composition of relations

- one-to-one correspondence between digraphs and binary relations
- arrow diagram for a binary relation *is* a directed graph

The **composition** of relations  $R$  and  $S$  on set  $A$  is denoted as  $S \circ R$ . Logically, this is what it means:

$$(a, c) \in S \circ R \iff \exists b : (b \in A \wedge (a, b) \in R \wedge (b, c) \in S)$$

Composition is applied *right to left*, much like composition of functions, or matrix transformations. Therefore,  $S \circ R$  means  $R$  is applied first, then  $S$ .



## 6.5 Graph powers and the transitive closure

A relation can be composed with itself. For example, consider relation  $P$ , which expresses parent-child relationship.

$xPy$  means  $x$  is the parent of  $y$ .

$xP \circ Pz$  means  $x$  is the grandparent of  $z$

A relation composed with itself also represents walks of different lengths.

$P \circ P$  represents all walks of length 2.

$P \circ P \circ P$  represents all walks of length 3.

### The Graph Power Theorem

: Let  $G$  be a directed graph. Let  $u$  and  $v$  be any two vertices in  $G$ . There is an edge from  $u$  to  $v$  in  $G^k$  if and only if there is a walk of length  $k$  from  $u$  to  $v$  in  $G$ .

$$R^1 = R$$

$$R^k = R \circ R^{k-1} \text{ for all } k \geq 2$$

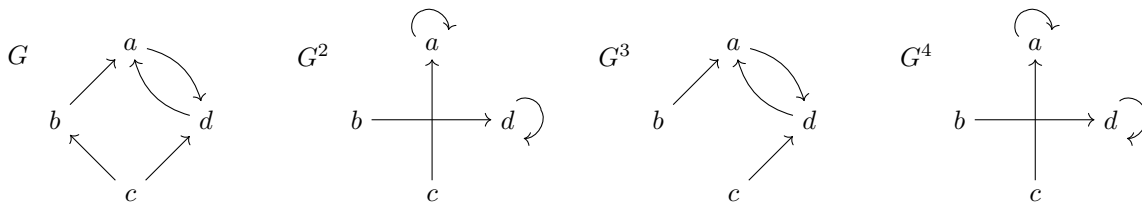
### Transitive Closure

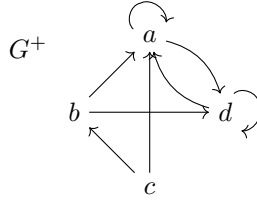
$$G^+ = G^1 \cup G^2 \cup G^3 \cup \dots$$

if  $G$  is not infinite, only up to the number of vertices are required for a complete graph of  $G^+$

$$G^+ = G^1 \cup G^2 \cup G^3 \cup \dots \cup G^{|V|}$$

Here is an example of a series of graph powers:



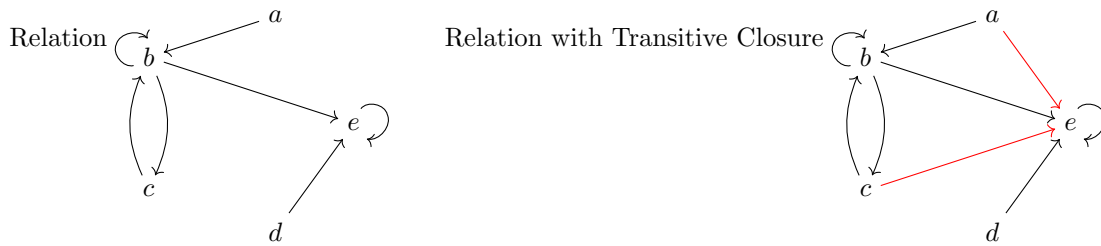


### Finding the Transitive Closure of a Relation $R$ on set $A$

Repeat until no pair is added to  $R$ .

If there are 3 elements  $x, y, z \in A$  such that  $xRy$  and  $yRz$  but not  $xRz$ , add  $xRz$  to  $R$ .

For example:



Edges added to find transitive closure are shown in red.

## 6.6 Matrix multiplication and graph powers

An  $n \times m$  **matrix** over set  $S$  is an array of elements from  $S$  with  $n$  rows and  $m$  columns. Each element in a matrix is called an *entry*. A **square matrix** has the same number of rows and columns. Here are a number of example matrixes

$$\begin{bmatrix} 1 & 3 \\ 3 & -5 \\ -2 & -2 \end{bmatrix}$$

$3 \times 2$  matrix over  $\mathbb{Z}$

$$\begin{bmatrix} 1.1 & 3.0 & -5.4 \\ -2.2 & -2.1 & 1 \end{bmatrix}$$

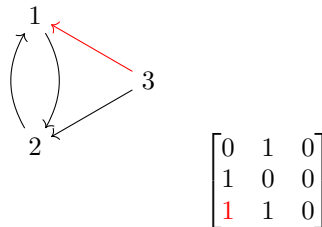
$2 \times 3$  matrix over  $\mathbb{Z}$

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$2 \times 2$  matrix over  $\{0, 1\}$

A directed graph  $G$  can be represented by a Matrix.

$n$  vertices  $\rightarrow n \times n$  matrix over the set  $\{0, 1\}$ , called an **adjacency matrix**



$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ \textcolor{red}{1} & 1 & 0 \end{bmatrix}$$

A **boolean matrix** is a matrix over the set  $\{0, 1\}$ , and boolean addition and multiplication are used. The **dot product** of a matrix  $A$  and  $B$  is defined only if  $\#$  of columns in  $A = \#$  of rows in  $B$

$$A = \begin{bmatrix} 1 & 1 & 1 \\ \textcolor{red}{1} & \textcolor{red}{0} & \textcolor{red}{1} \\ 0 & 1 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 & \textcolor{green}{1} \\ 0 & 0 & \textcolor{green}{1} \\ 1 & 0 & \textcolor{green}{1} \end{bmatrix}$$

$$\frac{\overset{1}{\times} \underset{1}{1}}{1} + \frac{\overset{0}{\times} \underset{1}{1}}{0} + \frac{\overset{1}{\times} \underset{1}{1}}{1} = 1 = (A \times B)_{2,3}$$

### Matrix Product

- denoted as  $AB$  or  $A \cdot B$
- uses a series of dot products to compute

There are a number of properties of matrix multiplication:

Commutative	$AB \neq BA$
Associative	$(AB)C = A(BC)$
Distributive	$A(B + C) = AB + AC$ $(B + C)A = BA + CA$
Multiplicative	$IA = A$ and $AI = A$ $OA = A$ and $AO = O$
Dimension	$(m \times n) \cdot (n \times k) = (m \times k)$

$k^{th}$  power of a matrix:

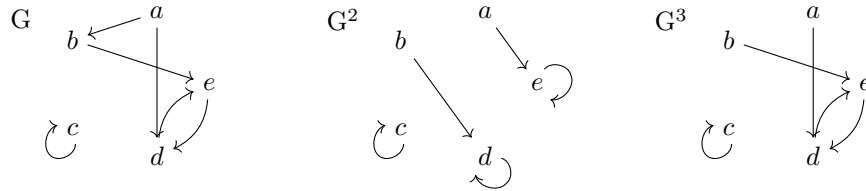
$$A^k = \underbrace{A \cdot A \cdots A}_{k \text{ times}}$$

If  $G$  is a digraph,  $G^k$  represents all walks of length  $k$  in  $G$ . There is an edge from vertex  $v$  to vertex  $w$  in  $G^k$  if and only if there is a walk of length *exactly*  $k$  from  $v$  to  $w$  in  $G$ . Matrix multiplication provides a systematic way of computing  $G^k$ .

1. Take *adjacency matrix*  $A$  for graph  $G$
2. Compute  $A^k$  by repeated *matrix multiplication*
3. Matrix  $A^k$  is the *adjacency matrix* for graph  $G^k$ .

### Relationship between powers of a graph and the powers of its adjacency matrix

Let  $G$  be a directed graph with  $n$  vertices and let  $A$  be the adjacency matrix for  $G$ . Then for  $k \geq 1$ ,  $A^k$  is the adjacency matrix of  $G^k$ , where boolean addition and multiplication are used to compute  $A^k$ .



Example:

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad A^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad A^3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

**Matrix Sum**

- denoted as  $A + B$
- well-defined if same # of row and # of columns

$$(A + B)_{i,j} = A_{i,j} + B_{i,j} \text{ for all } i \text{ and } j \text{ in range of } A \text{ or } B$$

For example,

$$\begin{array}{ccc} \begin{array}{c} \textcolor{blue}{0} \\ \begin{bmatrix} 1 & 0 & 1 \\ \textcolor{blue}{0} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ A \end{array} & + & \begin{array}{c} \textcolor{red}{1} \\ \begin{bmatrix} 0 & 0 & 1 \\ \textcolor{red}{1} & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \\ B \end{array} \\ \hline & = & \begin{array}{c} \textcolor{red}{1} \\ \begin{bmatrix} 1 & 0 & 1 \\ \textcolor{red}{1} & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \\ A + B \end{array} \end{array}$$

**Addition and Graph Union**

Let  $G$  and  $H$  be two directed graphs with the same vertex set. Let  $A$  be adjacency matrix for  $G$  and  $B$  the adjacency matrix for  $H$ .

Then the adjacency matrix for  $G \cup H = A + B$ , where boolean addition is used on the entries of matrices  $A$  and  $B$ .

$$\begin{array}{ccc} \begin{array}{c} \text{Digraph } G \\ \begin{array}{c} \text{Adjacency Matrix } A \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \end{array} & \begin{array}{c} \cup \\ + \end{array} & \begin{array}{c} \text{Digraph } H \\ \begin{array}{c} \text{Adjacency Matrix } B \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array} \end{array} \\ \hline & = & \begin{array}{c} \text{Digraph } G \cup H \\ \begin{array}{c} \text{Adjacency Matrix } A + B \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \end{array} \end{array} \end{array}$$

For digraph  $G$  and adjacency matrix  $A$  for  $G$ :

$$\begin{aligned} G^+ &= G^1 \cup G^2 \cup G^3 \cup \dots \cup G^n \\ A^+ &= A^1 + A^2 + A^3 + \dots + A^n \end{aligned}$$

**Condition for well-defined matrix multiplication**

$A_{m \times n} \times B_{s \times t}$  is only defined when  $n = s$ , and  $A \times B$  has dimensions  $m \times t$ . For example:

$$A_{3 \times 3} \begin{bmatrix} 0 & 2 & 3 \\ 1 & 0 & 1 \\ 2 & 0 & 1 \end{bmatrix} \times B_{3 \times 1} \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = (A \times B)_{3 \times 1} \begin{bmatrix} 5 \\ 4 \\ 0 \end{bmatrix}$$

**6.7 Partial orders**

A relation on set  $A$  is a **partial order** if it is:

- reflexive
- transitive



- anti-symmetric

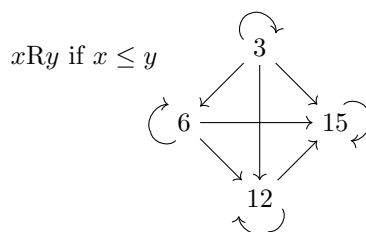
Notation:  $a \preceq b$  used to express  $aRb$

Domain along with a partial order defined on it is denoted  $(A, \preceq)$  and is called a **partial ordered set** or **poset**.

$\preceq \neq \leq$  (notice the curves)

Two elements of a poset are said to be *comparable* if  $x \preceq y$  or  $x \succeq y$ . Otherwise they are said to be *incomparable*. A partial order is a *total order* if every two elements in the domain are *comparable*.

Here is an example of a partial order:



- An element  $x$  is **minimal** element if there is no such  $y \neq x$  such that  $y \preceq x$
- An element  $x$  is **maximal** element if there is no such  $y \neq x$  such that  $x \preceq y$

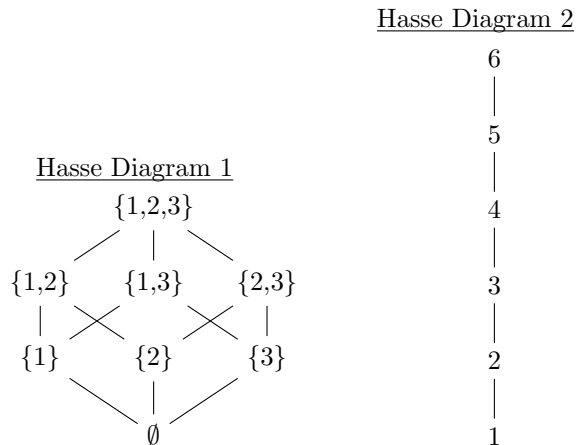
### Hasse Diagram

- useful way to depict a partial order on a finite set
- each element is represented by a point
- shows relationships by placing elements that are greater than others toward the top of the diagram.

### Rules for placement and for connecting segments

- if  $x \preceq y$ , then make  $x$  appear lower in the diagram than  $y$
- if  $x \preceq y$ , and there is no such  $z$  that  $x \preceq z \preceq y$ , then draw a segment connecting  $x$  and  $y$

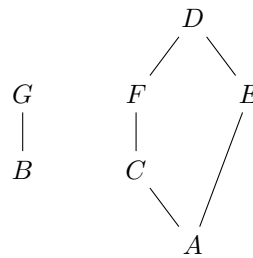
Examples: Hasse Diagrams for a partial order on the power set of  $\{1, 2, 3\}$ , and  $\{1, 2, 3, 4, 5, 6\}$ .



The first example uses a rule of  $A \preceq B \leftrightarrow A \subseteq B$ , while the second example uses a rule of  $x \preceq y \leftrightarrow x \leq y$ .

In general, if two elements are incomparable, then they are not connected at all by a path of line segments or the only paths between  $x$  and  $y$  require a change in direction from up to down or down to up. Consider this partial order on the set  $\{A, B, C, D, E, F, G\}$ :

Hasse Diagram 3



In this example,  $B \preceq G$ , and  $A \preceq D$ , but  $B$  and  $F$  are not comparable.

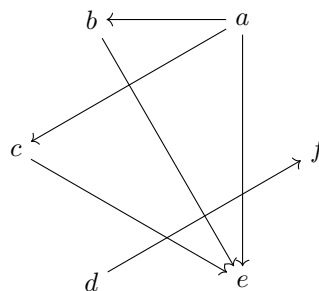
## 6.8 Strict orders and directed acyclic graphs

- Partial Order acts  $\preceq$  on a domain
- Strict Order acts  $\prec$  on a domain

A relation  $R$  is a **Strict Order** if  $R$  is *transitive*, *anti-symmetric*, and *anti-reflexive*.

- two elements are comparable if  $x \prec y$  or  $x \succ y$ , and otherwise incomparable
- strict order is a *total order* if every pair of elements is comparable
- element  $x$  is **minimal** if no  $y$  exists such that  $y \prec x$
- element  $x$  is **maximal** if no  $y$  exists such that  $x \prec y$

Here is an example of a strict order:



## 6.9 Equivalence relations

## 6.10 N-ary relations and relational databases

## 7 Computation

- 7.1 An introduction to algorithms
- 7.2 Asymptotic growth of functions
- 7.3 Analysis of algorithms
- 7.4 Finite state machines
- 7.5 Turing machines
- 7.6 Decision problems and languages

## 8 Induction and Recursion

- 8.1 Sequences
- 8.2 Recurrence relations
- 8.3 Summations
- 8.4 Mathematical induction
- 8.5 More inductive proofs
- 8.6 Strong induction and well-ordering
- 8.7 Loop invariants
- 8.8 Recursive definitions
- 8.9 Structural induction
- 8.10 Recursive algorithms
- 8.11 Induction and recursive algorithms
- 8.12 Analyzing the time complexity of recursive algorithms
- 8.13 Divide-and-conquer algorithms: Introduction and mergesort
- 8.14 Divide-and-conquer algorithms: Binary Search
- 8.15 Solving linear homogeneous recurrence relations
- 8.16 Solving linear non-homogeneous recurrence relations
- 8.17 Divide-and-conquer recurrence relations

## 9 Integer Properties

### 9.1 The Division Algorithm

### 9.2 Modular arithmetic

### 9.3 Prime factorizations

### 9.4 Factoring and primality testing

### 9.5 Greatest common factor divisor and Euclid's algorithm

### 9.6 Number representation

### 9.7 Fast exponentiation

### 9.8 Introduction to cryptography

### 9.9 The RSA cryptosystem

## 10 Introduction to Counting

- 10.1 Sum and Product Rules
- 10.2 The Bijection Rules
- 10.3 The generalized product rule
- 10.4 Counting permutations
- 10.5 Counting subsets
- 10.6 Subset and permutation examples
- 10.7 Counting by complement
- 10.8 Permutations with repetitions
- 10.9 Counting multisets
- 10.10 Assignment problems: Balls in bins
- 10.11 Inclusion-exclusion principle

## 11 Advanced Counting

### 11.1 Generating permutations

### 11.2 Binomial coefficients and combinatorial identities

### 11.3 The pigeonhole principle

### 11.4 Generating functions

## 12 Discrete Probability

- 12.1 Probability of an event
- 12.2 Unions and complements of events
- 12.3 Conditional probability and independence
- 12.4 Bayes' Theorem
- 12.5 Random variables
- 12.6 Expectation of random variables
- 12.7 Linearity of expectations
- 12.8 Bernoulli trials and the binomial distribution



## 13 Graphs

- 13.1 Introduction to Graphs
- 13.2 Graph representations
- 13.3 Graph isomorphism
- 13.4 Walks, trails, circuits, paths, and cycles
- 13.5 Graph connectivity
- 13.6 Euler circuits and trails
- 13.7 Hamiltonian cycles and paths
- 13.8 Planar coloring
- 13.9 Graph coloring

## 14 Trees

### 14.1 Introduction to trees

### 14.2 Tree application examples

### 14.3 Properties of trees

### 14.4 Tree traversals

### 14.5 Spanning trees and graph traversals

### 14.6 Minimum spanning trees