What is logic? And why on Earth is it relevant?

- **In philosophy:**
  Study of logic had begun by the ancient Greeks, whose educational system stressed the importance of competence in philosophy and rhetoric.
  Logic was used to formalise *correct argumentation* and *deduction*.
- **In mathematics:**
  The rules of logic are used to distinguish between correct and incorrect mathematical arguments (aka *proofs*).
  Logic also provides a precise foundation for mathematics.
- **In computer science, informatics, engineering:**
  - logic is used in the design of hardware (digital circuits)
  - logic is used in constructing computer programs (code)
  - logic is used in verifying the correctness of programs
  - logic is used in verifying the security of hardware and software
  - software systems have even been developed for constructing proofs automatically

Example: The need for software and hardware validation

Why do we need to verify the correctness of computer systems?

- The error in Intel's Pentium floating-point division unit — $ 500 000 000.
- Software mistake in the missile Ariane-5 — $ 500 000 000.
- Mistake in Denver's baggage handling system that postponed the opening of the new airport for 9 months — $ 1 100 000 per day.
- Something got wrong with the software on the Mars rover Spirit.
- What do you feel/say when the system you work with crashes?
- What about software controlling nuclear power plants, test-equipment in hospitals, airplanes, TVs, etc.?

In most designs, more time and effort is spent on validation than on construction.

However...

Studying logic is good, but you cannot have false hopes.

According to the famous German philosopher

Georg Wilhelm Friedrich **Hegel:**

Wissenschaft der Logik (1812-1816)

"Ganz so schlimm als der Metaphysik ist es der Logik nicht ergangen. Daß man durch sie denken lerne, was sonst für ihren Nutzen und damit für den Zweck derselben galt—gleichsam als ob man durch das Studium der Anatomie und Physiologie erst verdauen und sich bewegen lernen sollte—dieß Vorurtheil hat sich längst verloren, und der Geist des Praktischen dachte ihr wohl kein besseres Schicksal zu, als ihrer Schwester. Dessen ungeachtet, wahrscheinlich um einigen formellen Nutzens willen, wurde ihr noch ein Rang unter den Wissenschaften gelassen, ja sie wurde selbst als Gegenstand des öffentlichen Unterrichts beibehalten. Dieß bessere Loos betrifft jedoch nur das äußere Schicksal; denn ihre Gestalt und Inhalt ist derselbe geblieben, als er sich durch eine lange Tradition fortgeerbt, jedoch in dieser Ueberlieferung immer mehr verdünnt und abgemagert hatte; der neue Geist, welcher der Wissenschaft nicht weniger als der Wirklichkeit aufgegangen ist, hat sich in ihr noch nicht verspüren lassen. . . "

Logic no more teaches you to think

than physiology teaches you to digest

Basic building blocks of logic: propositions

A **proposition** is a **declarative sentence:** a sentence about which it can be meaningfully asked whether it is true (T) or false (F).

(Other notations: $\top$, 1, $+$ for true, $\bot$, 0, $-$ for false)

FOR EXAMPLE: *Today is Monday.*

$2 + 2 = 4$.

$2 + 2 = 5$.

All Martians like pepperoni on their pizza.

NOT DECLARATIVE: *Hi, how are you?*

Don't do that!

May the Force be with you.

$x + y = 4$.

NOT EVEN SENTENCE: *the president of the United States*

the pen in my hand

Propositional logic: the symbols we use

The area of logic that deals with propositions is called **propositional logic.**
It was first developed systematically more than 2300 years ago by the Greek philosopher **Aristotle.**

We use lower case letters (usually $p, q, r,$ or $p_1, p_2, \ldots$) to denote propositions whose structure we do not want to analyse any further.
We call these letters **propositional variables.**

We build compound expressions, called **formulas ,** denoting compound propositions. The formulas are built up from propositional variables using the following symbols:

- **logical connectives** $\neg,\ \wedge,\ \vee,\ \to,\ \leftrightarrow$
- parentheses (aka brackets) (and)

Compound propositions: negation

Let p be a proposition. The **negation of** p, denoted by $\boxed{\neg p}$, is the proposition **"It is not the case that p."**

(Other notations for negation: $\sim p$, $\overline{p}$)

The truth table of negation:

p	$\neg p$
T	F
F	T

FOR EXAMPLE:

p *Today is Tuesday.*

$\neg p$ *It is not that case that today is Tuesday.*

or simpler:

Today is not Tuesday.

or:

It is not Tuesday today.

Compound propositions: conjunction

Let p and q be propositions. The **conjunction of** p **and** q, denoted by $\boxed{p \wedge q}$, is the proposition "p **and** q."

(Other notations for conjunction: $p \,\&\, q$, $p \cdot q$)

The truth table of conjunction:

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

FOR EXAMPLE:

p *Today is Tuesday.*

q *It is raining today.*

$p \wedge q$ *Today is Tuesday and it is raining today.*

(BUT: in English $p \wedge q$ and $q \wedge p$ do NOT always mean the same
"Bob became sick and he visited the doctor."
"Bob visited the doctor and he became sick.")

Compound propositions: disjunction

Let p and q be propositions. The **disjunction** of p and q (also called as the **inclusive or** of p and q), denoted by $\boxed{p \vee q}$, is the proposition "p **or** q."

(Other notation for disjunction: $p + q$)

The truth table of disjunction (inclusive or):

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

FOR EXAMPLE:

Students who have taken either maths or physics A-levels are admitted.

BUT WHAT ABOUT THIS: *I'll go either to the west or to the east.*

The truth table of **exclusive or:**

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Compound propositions: conditional statements

Let p and q be propositions. The **conditional statement** $\boxed{p \to q}$ is the proposition **"if p then q."** A conditional statement is also called an **implication.**

Other readings of $p \to q$: **"q follows from p"** **"p implies q"**

(Other notations for implication: $p \Rightarrow q$, $p \supset q$)

The truth table of implication:

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

FOR EXAMPLE: *If I am elected, I'll lower taxes.*

If you solve all exercises in the exam, you will have an A.

If it is sunny today, we will go to the beach.

BUT WATCH OUT: seemingly 'weird' conditional statements can be true

If the Moon is made of green cheese then $2 \times 2 = 5$.

If the Moon is made of green cheese then $2 \times 2 = 4$.

If London is in the U.K. then $2 \times 2 = 4$.

Compound propositions: biconditional statements

Let p and q be propositions. The **biconditional statement** $\boxed{p \leftrightarrow q}$ is the proposition "p **if and only if** q."

A biconditional statement is also called a **bi-implication.**

Other readings of $p \leftrightarrow q$: "p **iff** q" "p **is necessary and sufficient for** q"

(Other notation for bi-implication: $p \Leftrightarrow q$)

The truth table of bi-implication:

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

WATCH OUT: implicit use of biconditional statements in English

If you finish your meal then you can have dessert. $p \to q$

what was really meant is $q \leftrightarrow p$:

You can have dessert if and **only if** *you finish your meal.*

Computing the truth tables of compound formulas

We can build compound formulas from propositional variables, using the logical connectives and the parentheses.

Given a compound formula $\boxed{(p \to (q \lor r)) \to ((p \land q) \to \neg r)}$, we do its truth table:

- It has $2^3 = 8$ rows, as the formula contains 3 propositional variables.
- Starting from 'inside out', we use a separate column to find the truth values of each compound formula that occurs in the building process. The truth values for the given formula are in the final column of the table.

p	q	r	$q \lor r$	$p \to (q \lor r)$	$p \land q$	$\neg r$	$(p \land q) \to \neg r$	$(p \to (q \lor r)) \to ((p \land q) \to \neg r)$
T	T	T	T	T	T	F	F	F
T	T	F	T	T	T	T	T	T
T	F	T	T	T	F	F	T	T
T	F	F	F	F	F	T	T	T
F	T	T	T	T	F	F	T	T
F	T	F	T	T	F	T	T	T
F	F	T	T	T	F	F	T	T
F	F	F	F	T	F	T	T	T

The precedence of logical operators

When do we **have to** use parentheses? Can we write something like $p \wedge q \vee r$?

Not really: $(p \wedge q) \vee r$ and $p \wedge (q \vee r)$ have **different** truth tables.
So in this case we **must** use brackets to specify what we mean.

However, to reduce the number of parentheses we still agree that

negation is 'stickier' than any other logical connective.

FOR EXAMPLE: $\neg p \wedge q$ means $(\neg p) \wedge q$

(so if we want to mean $\neg(p \wedge q)$, we **must** use brackets)

$(p \vee q) \rightarrow \neg r$ means $(p \vee q) \rightarrow (\neg r)$

$\neg\neg r$ means $(\neg(\neg r))$

Some books/lectures might introduce more precedence rules, but usually they are hard to remember.

RECOMMENDATION: apart from $\neg$

$\boxed{\text{please always use brackets to uniquely express what you mean!}}$

Propositional satisfiability, tautologies and contradictions

- A formula that is always true, no matter what the truth values of the propositional variables that occur in it, is called a **tautology** (aka **valid**). (So *every* row in the truth table of a tautology makes it true.)
- A formula that is always false is called a **contradiction.**
- A formula is called **satisfiable** if there is *at least one* row in its truth table that makes the formula true.
  (So a satisfiable formula is simply a formula that is NOT a contradiction.)

FOR EXAMPLE: The formula on lecture slide 11 is neither a tautology nor a contradiction, but it is satisfiable.

A simple tautology: $p \lor \neg p$

p	$\neg p$	$p \lor \neg p$
T	F	T
F	T	T

A simple contradiction: $p \land \neg p$

p	$\neg p$	$p \land \neg p$
T	F	F
F	T	F

Applications of propositional satisfiability

Problems in diverse areas can be modelled in terms of propositional satisfiability:

- in robotics
- in software testing
- in computer-aided design
- in machine vision
- in integrated circuit design
- in computer and other kinds of systems networks
- in bioinformatics and genetics
- in puzzles like Sudoku
- . . .

Consistency

A collection of formulas is called **consistent** if

- not only *each and every* formula in the collection is *satisfiable,*
- but also it is possible to find a truth value assignment to the propositional variables in them that makes *all* formulas in the collection true *at the same time.*

FOR EXAMPLE: Consider the following system specifications:

The diagnostic message is stored in the buffer or it is retransmitted. $p \lor q$

The diagnostic message is not stored in the buffer. $\neg p$

If the diagnostic message is stored in the buffer, then it is retransmitted. $p \to q$

p	q	$p \lor q$	$\neg p$	$p \to q$	
T	T	T	F	T	
T	F	T	F	F	
F	T	T	T	T	$\longleftarrow$
F	F	F	T	T	

Formalising English sentences

EXERCISE: Translate the following English sentences to propositional logic:

(1) *If I am not playing tennis, I am watching tennis.*

(2) *If I am not watching tennis, I am reading about tennis.*

(3) *I cannot do more than one of these activities at the same time.*

SOLUTION: First we choose our propositional variables.

(WATCH OUT: they **must** denote propositions! say t: "*playing tennis*" – no good)

t: "*I am playing tennis.*"

w: "*I am watching tennis.*"

r: "*I am reading about tennis.*"

Then we can formalise the above English sentences as follows:

(1) $\neg t \to w$

(2) $\neg w \to r$

(3) $\neg(t \wedge w) \wedge \neg(t \wedge r) \wedge \neg(w \wedge r)$

or $(t \wedge \neg w \wedge \neg r) \vee (\neg t \wedge w \wedge \neg r) \vee (\neg t \wedge \neg w \wedge r) \vee (\neg t \wedge \neg w \wedge \neg r)$

Logical equivalences

Two formulas that have the same truth values in all possible cases (that is, in all possible truth table rows) are called **logically equivalent.**

FOR EXAMPLE: $p \to q$ and $\neg p \vee q$ are logically equivalent

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

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

Notation: $p \equiv q$ denotes that p and q are logically equivalent formulas

(The symbol $\equiv$ is NOT a logical connective, $p \equiv q$ is just a shorthand.)

p and q are logically equivalent formulas iff the formula $p \leftrightarrow q$ is a tautology

How to show logical equivalence

EXERCISE 1: Show that $p \oplus q \equiv (p \vee q) \wedge \neg(p \wedge q)$

SOLUTION:

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

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

EXERCISE 2: Show that $p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$

SOLUTION:

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

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

A list of well-known logical equivalences

Equivalence	Name
$p \vee p \equiv p$ $p \wedge p \equiv p$	idempotent laws
$\neg\neg p \equiv p$	double negation law
$p \vee q \equiv q \vee p$ $p \wedge q \equiv q \wedge p$	commutative laws
$(p \vee q) \vee r \equiv p \vee (q \vee r)$ $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	associative laws
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	distributive laws
$\neg(p \vee q) \equiv \neg p \wedge \neg q$ $\neg(p \wedge q) \equiv \neg p \vee \neg q$	de Morgan's laws
$p \vee (p \wedge q) \equiv p$ $p \wedge (p \vee q) \equiv p$	absorption laws
$p \rightarrow q \equiv \neg q \rightarrow \neg p$	contraposition law

$\leadsto$ so it is OK to write $p \vee q \vee r$
and $p \wedge q \wedge r$

Logically correct arguments in propositional logic

An **argument** is a sequence of propositions:

$$\begin{array}{rll} & \left.\begin{array}{l} p_1 \\ p_2 \\ \vdots \\ p_n \end{array}\right\} & n \text{ **premises** (aka **assumptions**)} \\ \text{Therefore,} & \overline{\quad\quad\quad} & \\ & \quad q & \text{one **conclusion**} \end{array}$$

An argument is **logically correct** if

$\boxed{\text{in every situation that makes all premises true, the conclusion is true as well.}}$

a **situation** = a row in the truth table of $p_1, p_2, \dots, p_n, q$

= a possible assignment for the propositional variables in $p_1, p_2, \dots, p_n, q$

$\boxed{\begin{array}{l} p_1 \\ p_2 \\ \dots \\ p_n \\ \hline q \end{array}}$ is logically correct iff the formula $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \to q$ is a tautology

Logically correct argument: an example

- *If I am not playing tennis, I am watching tennis.* $\neg t \to w$
- *If I am not watching tennis, I am reading about tennis.* $\neg w \to r$
- *I cannot do more than one of these activities at the same time.* $\neg(t \wedge w) \wedge \neg(t \wedge r) \wedge \neg(w \wedge r)$

3 premises

Therefore,

I am watching tennis w conclusion

In every situation that makes all premises true, the conclusion is true as well:

	t	w	r	$\neg t \to w$	$\neg w \to r$	$\neg(t \wedge w) \wedge \neg(t \wedge r) \wedge \neg(w \wedge r)$	w
	T	T	T	T	T	F	T
	T	T	F	T	T	F	T
	T	F	T	T	T	F	F
	T	F	F	T	F	T	F
	F	T	T	T	T	F	T
→	F	T	F	$\boxed{\text{T}}$	$\boxed{\text{T}}$	$\boxed{\text{T}}$	$\boxed{\text{T}}$
	F	F	T	F	T	T	F
	F	F	F	F	F	T	F

premises conclusion

Incorrect argument: a simple example

If you solve every exercise in the textbook, then you will get an A. $s \to a$
You did not solve every exercise in the textbook. $\neg s$

Therefore,

You won't get an A. $\neg a$

Incorrect argument: It is not the case that
'in every situation that makes all premises true, the conclusion is true as well.'

So, to show that an argument is incorrect, it is **enough to find one situation** where

- all premises are true,
- but the conclusion is false.

	s	a	$s \to a$	$\neg s$	$\neg a$
	T	T	T	F	F
	T	F	F	F	T
$\longrightarrow$	F	T	$\boxed{\text{T}}$	$\boxed{\text{T}}$	$\boxed{\text{F}}$
	F	F	T	T	T

Rules of inference in propositional logic

We can always use a truth table to decide whether an argument is correct. However, this can be a **very** tedious approach: say, when the formulas in an argument include 10 different propositional variables, the truth table requires $2^{10} = 1024$ different rows.

Instead, we can first establish the correctness of some simple argument forms, called **rules of inference,** and then use them as building blocks to construct more complicated correct arguments.

FOR EXAMPLE: **modus ponens** (aka **rule of detachment**)

	$p \to q$	*If it snows today, then we will go skiing.*
	p	*It snows today.*
Therefore,		
	q	*We will go skiing.*

Some more rules of inference

$$\boxed{\begin{array}{l} p \to q \\ \neg q \\ \hline \neg p \end{array}}$$

modus tollens

$$\boxed{\begin{array}{l} p \to q \\ q \to r \\ \hline p \to r \end{array}}$$

chain of implications

$$\boxed{\begin{array}{l} p \to r \\ q \to r \\ \hline (p \vee q) \to r \end{array}} \qquad \boxed{\begin{array}{l} p \vee q \\ \neg p \\ \hline q \end{array}} \qquad \boxed{\begin{array}{l} p \\ q \\ \hline p \wedge q \end{array}} \qquad \boxed{\begin{array}{l} p \wedge q \\ \hline p \end{array}} \qquad \boxed{\begin{array}{l} p \\ \hline p \vee q \end{array}}$$

And we **always** have:

$p \equiv q$ iff both $\boxed{\begin{array}{c} p \\ \hline q \end{array}}$ and $\boxed{\begin{array}{c} q \\ \hline p \end{array}}$ are correct inference rules.

When formalising with propositional logic is clumsy

Consider the following argument:

Everyone in this class is feeling miserable today.

Therefore, ______________________________

Agi is feeling miserable today.

We can formalise this argument by choosing (a lot of) propositional variables

p_1	*Abby is feeling miserable today.*
	$\dots$
p_{42}	*Agi is feeling miserable today.*
	$\dots$
p_{185}	*Sandeep is feeling miserable today.*
	$\dots$
p_{247}	*Zoe is feeling miserable today.*

and then writing

$$\boxed{\begin{array}{l} p_1 \\ \dots \\ p_{42} \\ \dots \\ p_{185} \\ \dots \\ p_{247} \\ \hline p_{42} \end{array}}$$

When formalising with propositional logic is not possible

Every natural number is an integer number.

42 *is natural number.*

Therefore, $\overline{}$

42 *is an integer number.*

Formalising this argument in propositional logic

would require **infinitely many premises.**

$\rightsquigarrow$ we need a more powerful and expressive logic: **predicate logic**

Predicate logic basics: variables and predicates

Statements involving **variables,** such as

$$x > 3$$

$$x = y + 4$$

printer x *is networked to computer* y

are often found in maths texts, computer programs, and system specifications.

- These statements are neither true nor false when the values of the variables are not specified.
- Also, the variables x and y are NOT propositional variables: they do NOT denote propositions, but they denote **entities** (= things, objects, persons) such as: the number 42, the printer on my desk, or the president of the US.

We formalise "$x > 3$" as $\boxed{P(x)}$ where x is a variable, and P is the **predicate** (**property**) "*greater than* 3"

By assigning a **value** (a concrete entity) to its variable, $P(x)$ becomes a proposition that has a truth value:

$P(5)$ 5 *is greater than* 3. $\rightsquigarrow$ T

$P(1)$ 1 *is greater than* 3. $\rightsquigarrow$ F

More complex predicates

To formalise a statement like "$x = y + 4$" we need $\boxed{Q(x, y)}$ where

- x and y are variables, and
- Q is the predicate connecting two things (a **binary predicate**):
  "*the first number is the sum of the second number and* 4"

By assigning values to both of its variables, $Q(x, y)$ becomes a proposition that has a truth value:

$$\begin{aligned}
Q(1,5): &\quad 1 = 5 + 4. \quad \rightsquigarrow \quad \mathrm{F}\\
Q(7,3): &\quad 7 = 3 + 4. \quad \rightsquigarrow \quad \mathrm{T}\\
Q(5,1): &\quad 5 = 1 + 4. \quad \rightsquigarrow \quad \mathrm{T}\\
Q(6,6): &\quad 6 = 6 + 4. \quad \rightsquigarrow \quad \mathrm{F}
\end{aligned}$$

We use upper case letters (usually P, Q, A, B or $P_1, P_2 \dots$) to denote predicates.

Predicate logic basics: the universal quantifier

By assigning a concrete entity to x, the expression $P(x)$ becomes a proposition.

Quantification is another important way of creating a proposition from $P(x)$. Quantification expresses the extent to which a predicate is true over a range of entities, called the **domain**.

The **universal quantification** of $P(x)$ for a particular domain, denoted by $\boxed{\forall x\, P(x)}$, is the proposition "$P(x)$ **is true for all values of** x **from the domain.**"

- The truth value of $\forall x\, P(x)$ might change when we change the domain.
- The domain must always be specified when a universal quantifier is used.

FOR EXAMPLE: Suppose $P(x)$ is "$x > 0$".
If the domain is the positive integers, then $\forall x\, P(x)$ is true.
If the domain is all integers, then $\forall x\, P(x)$ is false.

EXERCISE: Suppose $P(x)$ is "$x^2 > 0$".
Show that $\forall x\, P(x)$ is false, when the domain is all integers.

SOLUTION: It is enough to find a **counterexample:** an entity in the domain for which $P(x)$ is false. As $0^2 > 0$ is false, $x = 0$ is such a counterexample.

Rules of inference involving the universal quantifier

- <u>**universal instantiation:**</u> for each and every entity c in the domain we have a correct inference rule of the form

$$\boxed{\dfrac{\forall x\, P(x)}{P(c)}}$$

FOR EXAMPLE: *Everyone in this class is feeling miserable today.*

Therefore, ________________________________

Agi is feeling miserable today.

- <u>**universal generalisation:**</u>

$$\boxed{\dfrac{P(c) \ \text{ for any arbitrary entity } c \text{ in the domain}}{\forall x\, P(x)}}$$

We will return to this inference rule later with examples.

Predicate logic basics: the existential quantifier

The **existential quantification** of $P(x)$ for a particular domain, denoted by $\boxed{\exists x\, P(x)}$, is the proposition "**there exists a value for x in the domain such that $P(x)$ is true.**"

- The truth value of $\exists x\, P(x)$ might change when we change the domain.
- The domain must always be specified when an existential quantifier is used.

FOR EXAMPLE: Suppose $P(x)$ is "$x < -3$".

- If the domain is all integers, then $\exists x\, P(x)$ is true.
  WHY? because, say, if $x = -5$ then $P(-5)$ is true.
  (It is enough to give an **example.**)
- If the domain is the positive integers, then $\exists x\, P(x)$ is false.

More inference rules involving quantifiers

- **existential generalisation:** for every entity c in the domain we have a correct inference rule of the form

$$\boxed{\frac{P(c)}{\exists x\, P(x)}}$$

FOR EXAMPLE: *Marla is reading the lecture notes.*

Therefore, ______________________________

There is someone in the class reading the lecture notes.

- **de Morgan laws for quantifiers:**

$$\boxed{\frac{\neg\forall x\, P(x)}{\exists x\, \neg P(x)}} \qquad \boxed{\frac{\exists x\, \neg P(x)}{\neg\forall x\, P(x)}} \qquad \boxed{\frac{\neg\exists x\, P(x)}{\forall x\, \neg P(x)}} \qquad \boxed{\frac{\forall x\, \neg P(x)}{\neg\exists x\, P(x)}}$$

Theorems and proofs

In maths when a proposition is such that

- it can be shown to be true, and
- it is important enough,

then it is called a **theorem**.

A **proof of a theorem t** is a sequence of propositions ending with t

$$\boxed{\begin{array}{l} \left.\begin{array}{l} p_1 \\ p_2 \\ \vdots \\ p_n \end{array}\right\} \text{premises} \\ \hline t \end{array}}$$

such that *each and every* of the premises is

- either an **axiom** (= a proposition whose truth we simply assume),
- or can be obtained from earlier premises in the sequence by a logically correct argument.

In this case, we also say that **theorem t follows from the axioms.**

Understanding how theorems are stated

Many theorems assert that a property holds for **all entities in a domain,** such as the integers or the natural numbers.

Although the precise statement of such theorems needs to include a **universal quantifier,** the standard convention in mathematics is to omit it.

FOR EXAMPLE: The statement

"*If* $x > 5$*, where* x *is a positive integer, then* $x^2 > 30$*.*"

really means

"*For all positive integers* x*, if* $x > 5$ *then* $x^2 > 30$*.*"

Finding proofs is a tricky business

There are statements about which most people suspect that they true,
but so far nobody could actually find a proof for them.
These kinds of statements are called in maths **conjectures.**

EXAMPLE 1: the **Goldbach conjecture**

Every even integer greater than 2 can be expressed as the sum of two primes.

EXAMPLE 2: the **Collatz conjecture**

A sequence of numbers is defined as follows: We start with any positive integer, our initial value. Then the next number is obtained from the previous number by following one of two cases:

- If the previous number is even, the next number is half the previous number.
- If the previous number is odd, the next number is 3 times the previous number plus 1.

The conjecture is that *no matter what is our initial value, we always reach* 1.

(For instance, starting with 12, we get the sequence $12, 6, 3, 10, 5, 16, 8, 4, 2, 1$.)

Proof techniques basics: general direct proofs

This technique is used to prove theorems of the form $\forall x\,\big(A(x) \to B(x)\big)$:

- First, we choose an **arbitrary** entity from the domain. It is important that this entity is indeed arbitrary, meaning we do not use any specifics about it.
  No matter what is the domain, we need to use some 'pointer' to refer to this chosen entity. We can denote it with whatever symbol we want: $c, x, \star, \smiley, \ldots$
- Then, we prove the conditional statement $A(\smiley) \to B(\smiley)$ for this arbitrary $\smiley$ by a **direct proof:**

  We **assume** that $A(\smiley)$ is true, and use axioms, definitions, and previously proven theorems, together with rules of inference, to show that then $B(\smiley)$ must also be true.

- Then we apply the rule of **universal generalisation:** (cf. lecture slide 30)

$$\boxed{\dfrac{A(\smiley) \to B(\smiley) \quad \text{for any } \smiley}{\forall x\,\big(A(x) \to B(x)\big)}}$$

FOR EXAMPLE: Theorem: *"If n is an odd integer, then n^2 is odd."*

Proof: Let n be an arbitrary integer. Assume that n is odd. By the definition of being odd, $n = 2k{+}1$ for some integer k. So $n^2 = (2k+1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$. As $2k^2 + 2k$ is an integer, by the definition of being odd, it follows that n^2 is odd.

Proof techniques basics: indirect proofs

EXERCISE: Show that at least 4 of any 22 dates in the calendar must fall on the same day of the week.

SOLUTION: **Proof by contradiction.** As a proof by contradiction does not prove a result `directly', it is also called an **indirect proof.**

Let p be: "*At least* 4 *of* 22 *chosen dates fall on the same day of the week.*"

Suppose that $\neg p$ is true. This means:

"*At most* 3 *of* 22 *chosen dates fall on the same day of the week.*"

Because there are 7 days of the week, this implies that at most 21 dates could have been chosen, as for each of the 7 days of the week, at most 3 of the chosen dates could fall on that day. So, if q is the statement "22 *dates are chosen*" then we have shown that $\neg p \to \neg q$ is true. So, using q as a premise, we reached a contradiction: we showed that $(q \wedge \neg p) \to (q \wedge \neg q)$ is true.

WHAT? $q \wedge \neg q$ is a contradiction: it is false whatever proposition q is.

So the only way $(q \wedge \neg p) \to (q \wedge \neg q)$ can be true is when $q \wedge \neg p$ is false.

As q is true (being our premise), the only way $q \wedge \neg p$ can be false

is when $\neg p$ is false, and so p is true.

Mistakes in proofs: an example

EXERCISE: What is wrong with this supposed "proof" that $1 = 2$?

"*Proof:*" We use these steps, where a and b are two equal positive integers:

(1)	$a = b$	axiom
(2)	$a^2 = ab$	multiply both sides of (1) by a
(3)	$a^2 - b^2 = ab - b^2$	subtract b^2 from both sides of (2)
(4)	$(a - b)(a + b) = b(a - b)$	factor both sides of (3)
(5)	$a + b = b$	divide both sides of (4) by $a - b$
(6)	$2b = b$	replace a by b in (5) because $a = b$ and simplify
(7)	$2 = 1$	divide both sides of (6) by b

SOLUTION:

Every step is correct except for (5) from (4): we divided both sides by $a - b$. The error is that $a - b$ equals 0, and division of both sides of an equation by the same quantity is correct as long as this quantity is **not** 0.

Mistakes in proofs II: circular reasoning

This kind of mistake occurs when one or more steps of a “proof” are based on the truth of the statement being proved.

EXERCISE: What is wrong with this supposed “proof” of the (true) statement “*if* n^2 *is even, then* n *is even*” ?

“*Proof:*” Suppose that n^2 is even. Then $n^2 = 2k$ for some integer k. Let $n = 2l$ for some integer l. This shows that n is even.

SOLUTION: The statement “let $n = 2l$ for some integer l” occurs in the “proof.” But no argument has been given to show that n can be written as $2l$ for some integer l . (In fact, this is precisely what we need to prove.)

Proof techniques basics: proof by induction

To prove that a property $P(n)$ is true for all positive integers n,
we complete two steps:

Basis step: We prove that $P(1)$ is true.

Inductive step: We prove that the conditional statement $P(k) \to P(k+1)$ is true for all positive integers k.

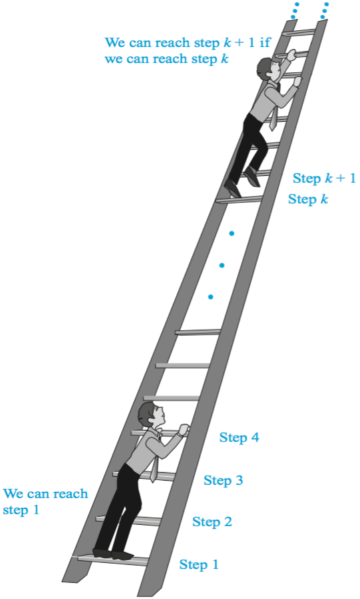
To complete the inductive step of a proof by induction,

- we **assume** that $P(k)$ is true for an **arbitrary** positive integer k,
- and show that **under this assumption** $P(k+1)$ must also be true.

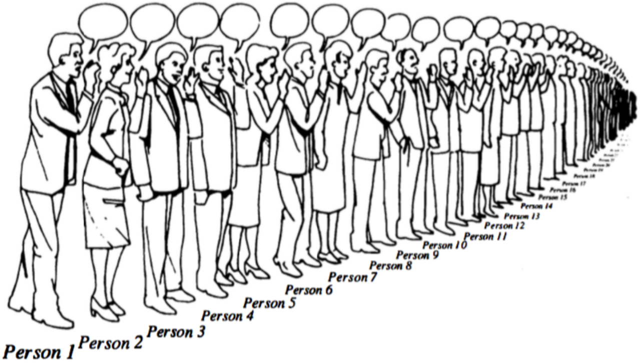
The assumption that $P(k)$ is true is called the **inductive hypothesis (IH).**

IMPORTANT! A proof by induction is **not** circular reasoning.
In a proof by induction it is **not** assumed that $P(k)$ is true for all positive integers.
It is only shown that **if it is assumed** that $P(k)$ is true, **then** $P(k+1)$ is also true.

Ways to remember proof by induction



climbing an infinite ladder



people telling secrets

Proof by induction: an example

EXERCISE: Show that for every positive integer n,

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

SOLUTION: **Basis step:** $P(1)$ is true, because $1 = \frac{1(1+1)}{2}$.

Inductive step: For the inductive hypothesis we assume that $P(k)$ holds for an arbitrary positive integer k. That is, we assume that

$$1 + 2 + \cdots + k = \frac{k(k+1)}{2}. \qquad \textbf{(IH)}$$

Under this assumption, it must be shown that $P(k+1)$ is true. So let's see:

$$\boxed{1 + 2 + \cdots + (k+1)} = (1 + 2 + \cdots + k) + (k+1) \overset{\text{by IH}}{=} \frac{k(k+1)}{2} + (k+1) =$$

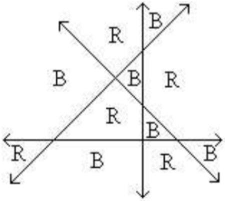
$$= \frac{k(k+1) + 2(k+1)}{2} = \frac{(k+2)(k+1)}{2} = \boxed{\frac{(k+1)\big((k+1)+1\big)}{2}}$$

Proof by induction: another example

There is a famous theorem called the **Four Colour Theorem.** It states that any map can be coloured with four colours such that any two adjacent countries (which share a border, but not just a point) must have different colours.
This theorem was **very** difficult to prove.

The **Simple Two Colour Theorem:** We consider only **simple maps**, when we divide the plane into regions by drawing **straight lines.** The theorem says that any simple map can be coloured using no more than **two colours** (say, Red and Blue) such that no two regions that share a border (not just a point) have the same colour.

Here is an example of a two-coloured simple map:



(fact) If we swap Red ↔ Blue in ANY two-colouring, we still have a two-colouring.

Let $\boxed{P(n)}$ be the statement: *Every simple map with n lines can be two-coloured.*

The theorem says: $P(n)$ is true for every positive integer n.

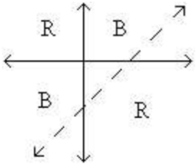
Proving the Simple Two Colour Theorem by induction

Basis step: $P(1)$ is true, because every simple map with 1 line can clearly be two-coloured, with Red on one side of the line, and Blue the other.

Inductive step: Assume that $P(k)$ is true, that is, any simple map with k lines can be two-coloured (this is the **IH**).

Under this assumption, it must be shown that $P(k+1)$ is true. So let's see:
We are given a map with $k+1$ lines. Let's remove one line L. We obtain a map with just k lines, so by the **IH,** we can two-colour it.
Now let us place back the line L we removed, and

- leave colours on one side of line L unchanged
- on the other side of line L, swap Red ↔ Blue.



We claim that this is a valid two-colouring for the given map with $k+1$ lines:

- Regions which do not border L are properly two-coloured by **(fact)**.
- But what about regions that do share a border with L? We must be certain that any two such regions have opposite colouring. But any two regions that border L must have been the same region when L was removed, so the reversal of colours on one side of L guarantees an opposite colouring.

Mistakes in proof by induction

Here is **George Pólya**'s famous "proof" by induction of the statement
"All horses have the same colour.":

$P(n)$: All horses in any group of n horses have the same colour.

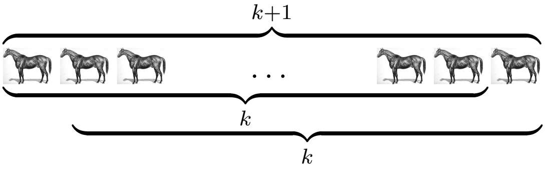
Basis step: $P(1)$ is clearly true.

Inductive step: Assume that $P(k)$ is true, that is, all horses in any group of k horses have the same colour (this is the **IH**).

Under this assumption, it must be shown that $P(k+1)$ is true. So let's see:
We are given $k+1$ horses.

- First, exclude the last horse and look only at the first k horses. By the **IH,** all horses in this group are of the same colour.
- Next, exclude the first horse and look only at the last k horses. By the **IH,** all horses in this group are of the same colour.
- These two groups overlap in the middle, hence all $k+1$ horses are of the same colour.

$\boxed{\textbf{??}}$



Sets: what is a set?

To communicate, we sometimes need to agree on the meaning of certain terms. If the same idea is mentioned several times in a discussion, we often replace it with some new terminology and shorthand notation.

A **set** is a collection of things, any things, called its **elements**.

If S is a set and x is an element in S, then we write

$$\boxed{x \in S}.$$

If x is **not** an element in S, then we write

$$\boxed{x \notin S}.$$

If **both** $x \in S$ **and** $y \in S$, we often denote this fact by the shorthand notation

$$\boxed{x, y \in S}.$$

Describing sets by listing

To describe a set we need to describe its elements in some way.

- The simplest way to describe a set is to **explicitly name its elements**.

  FOR EXAMPLE: We can form a set A by collecting three 'things':
  Cinderella, *Tasmania* and *Tuesday*.

  Any set defined this way is denoted by listing its elements, separated by commas, and surrounding the listing with braces:

  $$\boxed{A = \{\textit{Cinderella}, \textit{Tasmania}, \textit{Tuesday}\}}\,.$$

- **Anything** can be an element of a set.

  In particular, a set can be an element of another set.

  FOR EXAMPLE: The set $B = \{x, \{x, y\}\}$ has two elements:

  - one element is x ,
  - and the other element is the set $\{x, y\}$.

  (So we can write $x \in B$ and $\{x, y\} \in B$.)

Important features of sets

The only thing that matters about a set:

what is **IN** it and what is **NOT IN** it.

- **Repeated occurrences of elements don't matter:**

  $\{H, E, L, L, O\}$, $\{H, H, H, E, L, L, O\}$, and $\{H, E, L, O\}$ describe the **same** set (namely, the set having four elements: the letters H, E, L, and O).

  So it is best to be economical and list everything only once: $\{H, E, L, O\}$.

- **The order of listing doesn't matter either:**

  $\{H, E, L, O\}$, $\{E, H, L, O\}$, $\{E, H, O, L\}$, and $\{O, L, E, H\}$

  all describe the **same** set.

Special sets

- The set with no elements is called the **empty set**. The empty set is denoted by $\{\ \}$ or more often by the symbol

  $$\boxed{\emptyset}\,.$$

  The empty set has no elements. So no matter what thing x denotes, $x \notin \emptyset$.

  On the other hand, the empty set can be an element of another set.

  FOR EXAMPLE: $\emptyset \in \{a, \emptyset, \mathit{Joe}\}$ or $\emptyset \in \{\emptyset\}$.

- Any set with one element is called a **singleton**.

  FOR EXAMPLE: $\{a\}$ and $\{\mathit{Monday}\}$ are singletons,

  but $\{x, \emptyset\}$ is not.

Equality of sets

Two sets are **equal** if they have the same elements.

We denote the fact that two sets A and B are equal by writing

$$\boxed{A = B}.$$

If the sets A and B are **not equal,** we write

$$\boxed{A \neq B}.$$

FOR EXAMPLE:

$$\{u, g, h\} = \{h, u, g\}$$

$$\{a, b, c\} \neq \{a, c\}$$

$$\{\textit{Monday}\} \neq \emptyset$$

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

$$\{2\} \neq \{\{2\}\}$$

Finite and infinite sets

Suppose we start counting the elements of a nonempty set S, one element per second. If a point of time is reached when all the elements of S have been counted (we might need to have one of our descendants to finish up), then we say that S is **finite**.

If the counting never stops, then S is an **infinite** set.

EXAMPLES OF INFINITE SETS:

Z $= \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$ — **integers**

N $= \{0, 1, 2, 3, \dots\}$ — **natural numbers**

N$^+$ $= \{1, 2, 3, \dots\}$ — **positive natural numbers** $=$ **positive integers**

odd natural numbers

Q — the set of **rational numbers:**

any number that can be expressed as $\dfrac{m}{n}$ for some integers m and $n \neq 0$

R — the set of **real numbers:**

they include all rational numbers, and

also **irrational numbers** like π, $\sqrt{2}$, $\dots$

R$^+$ — the set of **positive real numbers**

Describing sets by properties: the set builder notation

Describing an infinite set by listing, using $\ldots$, might sometimes be possible (like we did with **N**), but even then a bit imprecise.

But listing, say, rational numbers can be quite tricky. Instead of listing elements, we can describe a property that the elements of the set satisfy.

If $\mathcal{P}$ is a property, then the set S whose elements have property $\mathcal{P}$ is denoted by

$$\boxed{S \;=\; \{x \mid x \text{ has property } \mathcal{P}\}}\,.$$

We read this as

"S **is the set of all** x **such that** x **has property** $\mathcal{P}$**.**"

Or, if we also know that all the elements in S come from a larger set A, then we can write

$$\boxed{S = \{x \in A \mid x \text{ has property } \mathcal{P}\}}\,.$$

Describing sets by properties: an example

Let *Odd* be the set of all odd integers. Then we can describe *Odd* in several ways:

$$\begin{aligned} \textit{Odd} &= \{\dots, -5, -3, -1, 1, 3, 5, \dots\} \\ &= \{x \mid x \text{ is an odd integer}\} \\ &= \{x \in \mathbf{Z} \mid x \text{ is odd}\} \\ &= \{x \mid x = 2k+1 \text{ for some integer } k\} \\ &= \{x \mid x = 2k+1 \text{ for some } k \in \mathbf{Z}\} \end{aligned}$$

We can also use more complex expressions on the left hand side of $\mid$ in the set builder notation:

$$\begin{aligned} \textit{Odd} &= \{2k+1 \mid k \text{ is an integer}\} \\ &= \{2k+1 \mid k \in \mathbf{Z}\} \end{aligned}$$

Russell's paradox

Not every property is suitable for describing sets. Here is a tricky one:

Let T be the **set** of all sets that are not elements of themselves:

$$\boxed{T = \{A \mid A \text{ is a set and } A \notin A\}}$$

<u>QUESTION: Is T an element of T or not ?</u>

Well, either $T \in T$ or $T \notin T$. Let us examine both cases:

- If $\underline{T \in T}$, then the property we used to describe T must hold for T. But then $\underline{T \notin T}$. So this case is impossible.

- If $\underline{T \notin T}$, then the property we used to describe T does not hold for T. So it is not the case that "T is a set and $T \notin T$."
  So, by de Morgan's law in logic, either T is not a set, or $T \in T$.
  But as T is a set, $\underline{T \in T}$ follows. So this case is impossible as well.

Describing sets by recursion

A recursive description of a set S consists of three steps:

(1) **Basis step:**
Specify one or more elements of S.

(2) **Recursive step:**
Give one or more rules to construct new elements of S from existing elements of S.

(3) **Exclusion rule:**
State that S consists **only** of the elements that are specified by the basis step, or generated by successive applications of the recursive step.
Nothing else is in S.
(This step is usually assumed rather than stated explicitly.)

Describing sets by recursion: examples

- The set **N** of natural numbers can be defined recursively:

  Basis step: $0 \in$ **N**

  Recursive step: If $n \in$ **N** then $n + 1 \in$ **N**.

- The set *Odd* of odd integers can be defined recursively:

  Basis step: $813 \in$ *Odd*

  Recursive step: If $n \in$ *Odd* then $n + 2 \in$ *Odd* and $n - 2 \in$ *Odd*.

- The set $A = \{3k + 1 \mid k \in \mathbf{N}\}$ can be defined recursively:

  Basis step: $1 \in A$

  Recursive step: If $x \in A$ then $x + 3 \in A$.

- The set F of all formulas of propositional logic can be defined recursively:

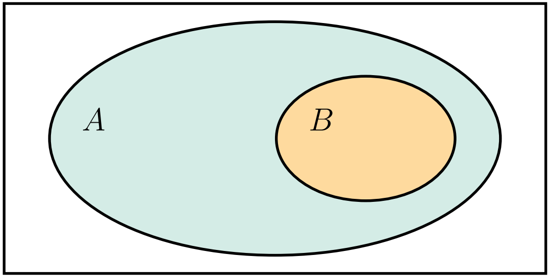
  Basis step: every propositional variable is in F

  Recursive step: If $p \in F$ and $q \in F$, then $(\neg p) \in F$,
  $(p \wedge q) \in F$, $(p \vee q) \in F$, $(p \rightarrow q) \in F$ and $(p \leftrightarrow q) \in F$.

Subsets

A set B is a **<u>subset</u>** of a set A, if every element of B is also an element of A.

Notation: $\boxed{B \subseteq A}$.



Venn diagram of $B \subseteq A$

FOR EXAMPLE: $\{a, c\} \subseteq \{a, b, c, d\}$ $\{0, 1, 5\} \subseteq \mathbf{N}$ $\mathbf{N} \subseteq \mathbf{Z}$ $\mathbf{N} \subseteq \mathbf{N}$

- We **always** have, for every set S: $\boxed{S \subseteq S}$ $\boxed{\emptyset \subseteq S}$
- If B is a subset of A, and there is some element in A that is **not** in B, then we say that <u>B is a **proper subset** of A,</u> and write $\boxed{B \subset A}$.
  In other words, $B \subset A$ if $B \subseteq A$ but $B \neq A$.

How to prove that $A \subseteq B$ or $A \subset B$?

Let $A = \{x \mid x$ is a prime number and $42 \leq x \leq 51\}$,

$B = \{x \mid x = 4k + 3$ and $k \in \mathbf{N}\}$.

<u>EXERCISE 1:</u> Show that $A \subseteq B$.

SOLUTION: We need to show that **every** element in A is also in B.

This is a statement of the form $\forall x\,(\text{“}x \in A\text{”} \to \text{“}x \in B\text{”})$.

Take an **arbitrary** $x \in A$. Then x is a prime number and $42 \leq x \leq 51$.

So either $x = 43$ or $x = 47$.

- We can have $43 = 4 \cdot 10 + 3$. So the choice of $k = 10$ shows that $\underline{43 \in B}$.
- We can have $47 = 4 \cdot 11 + 3$. So the choice of $k = 11$ shows that $\underline{47 \in B}$ as well.

Therefore, we have shown that $A \subseteq B$.

<u>EXERCISE 2:</u> Show that $A \subset B$.

SOLUTION: We have just shown that $A \subseteq B$. So it remains **to find an element** $x \in B$ such that $x \notin A$. For example, $x = 3$ is such:

- As $0 \in \mathbf{N}$ and $3 = 4 \cdot 0 + 3$, we have $\underline{3 \in B}$.
- On the other hand, $42 \not\leq 3$, so we have $\underline{3 \notin A}$.

How to prove that $A \not\subseteq B$?

Let $A = \{3k+1 \mid k \in \mathbf{N}\}$,

$B = \{4k+1 \mid k \in \mathbf{N}\}$.

<u>Exercise:</u> Show that $A \not\subseteq B$, that is, A is **not** a subset of B.

Solution: This is a statement of the form $\neg\forall x \, (\text{“}x \in A\text{”} \to \text{“}x \in B\text{”})$.

So we need **to find a counterexample:** an element $x \in A$ such that $x \notin B$.

For example, $x = 4$ is such:

- $4 = 3 \cdot 1 + 1$. So $k = 1$ shows that $\underline{4 \in A}$.
- We need to show that $4 \notin B$, that is, there is **no** $k \in \mathbf{N}$ such that $4 = 4k+1$.
  Let's see. If $k = 0$, then $4k+1 = 4 \cdot 0 + 1 = 1 \neq 4$.
  And if $k \geq 1$, then $4k+1 \geq 4 \cdot 1 + 1 = 5 > 4$.
  So it is not possible to find a $k \in \mathbf{N}$ such that $4 = 4k+1$, and so $\underline{4 \notin B}$.

Therefore, we have shown $A \not\subseteq B$.

How to prove $A = B$ or $A \neq B$?

$\boxed{A = B}$ means that sets A and B have the **same elements.**

This is a statement of the form $\forall x\,\big(\text{“}x \in A\text{”} \leftrightarrow \text{“}x \in B\text{”}\big)$.

$$\boxed{A = B \qquad \text{iff} \qquad A \subseteq B \qquad \text{and} \qquad B \subseteq A}$$

$$\forall x\,\big(\text{“}x \in A\text{”} \to \text{“}x \in B\text{”}\big) \;\land\; \forall x\,\big(\text{“}x \in B\text{”} \to \text{“}x \in A\text{”}\big)$$

$$\boxed{A \neq B \qquad \text{iff} \qquad A \not\subseteq B \qquad \text{or} \qquad B \not\subseteq A}$$

$$\exists x\,\big(\text{“}x \in A\text{”} \land \text{“}x \notin B\text{”}\big) \;\lor\; \exists x\,\big(\text{“}x \in B\text{”} \land \text{“}x \notin A\text{”}\big)$$

So:

- If the task is to show that $\boxed{A = B}$,

  then we need to show **BOTH** $\boxed{A \subseteq B}$ **AND** $\boxed{B \subseteq A}$.

- If the task is to show that $\boxed{A \neq B}$,

  then we **EITHER** need **to find an element** in A that is not in B,

  OR to find an element in B that is not in A.

The power set of a set

The **set of all subsets** of a set S is called the **power set of S** which we denote by

$$\boxed{P(S)}\,.$$

So $P(S) = \{A \mid A \subseteq S\}$.

FOR EXAMPLE:

- As for EVERY set S, we always have $\emptyset \subseteq S$ and $S \subseteq S$,

  we always have $\boxed{\emptyset \in P(S)}$ and $\boxed{S \in P(S)}$.

- $P\big(\{u,v,w\}\big) = \big\{\emptyset, \{u\}, \{v\}, \{w\}, \{u,v\}, \{u,w\}, \{v,w\}, \{u,v,w\}\big\}$

  WATCH OUT! $\{\emptyset\} \notin P\big(\{u,v,w\}\big)$

- $P\big(\{\textit{Joe}, \textit{Tuesday}\}\big) = \big\{\emptyset, \{\textit{Joe}\}, \{\textit{Tuesday}\}, \{\textit{Joe}, \textit{Tuesday}\}\big\}$

- $P(\mathbf{N}) = \{X \mid X \subseteq \mathbf{N}\}$. So, say, $\{453, 11, 5\} \in P(\mathbf{N})$, $\{81, 2\} \in P(\mathbf{N})$, $\emptyset \in P(\mathbf{N})$,

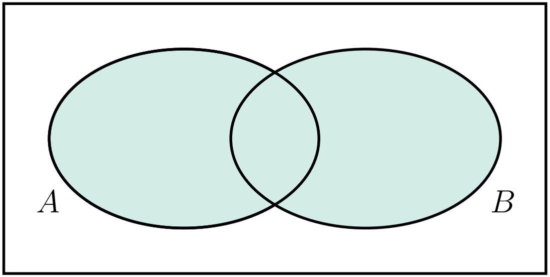
  but $\{3, -1\} \notin P(\mathbf{N})$, $\{\frac{5}{2}, 23\} \notin P(\mathbf{N})$, $\{\emptyset\} \notin P(\mathbf{N})$.

Set operations: union

The **union** of sets A and B is the set

$$\boxed{A \cup B \;=\; \{x \mid x \in A \;\text{ (inclusive) or }\; x \in B\}}\,.$$

$A \cup B$ consists of those elements that are either in A, or in B, or in both.



Venn diagram of $A \cup B$

FOR EXAMPLE: $A = \{4, 7, 8\}$ and $B = \{10, 4, 9\}$.

Then $A \cup B = \{4, 7, 8, 9, 10\}$.

Properties of union

- $A \cup \emptyset = A$ — **identity law**
- $A \cup B = B \cup A$ — **commutative law**
- $(A \cup B) \cup C = A \cup (B \cup C)$ — **associative law**

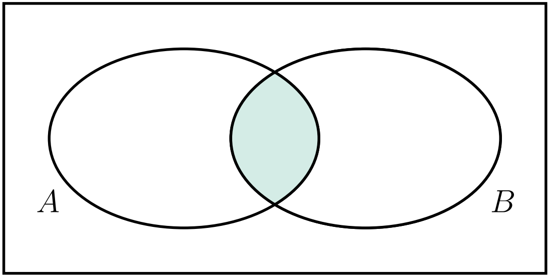
  So we can meaningfully write, say, $A \cup B \cup C \cup D \cup E$ or $\bigcup_{i=1}^{n} A_i$.
- $A \cup A = A$ — **idempotent law**
- $A \subseteq B$ iff $A \cup B = B$

Set operations: intersection

The **intersection** of sets A and B is the set

$$\boxed{A \cap B \ = \ \{x \mid x \in A \ \text{ and } \ x \in B\}}\,.$$

$A \cap B$ consists of those elements that are both in A and in B.



Venn diagram of $A \cap B$

FOR EXAMPLE: $A = \{4, 7, 8\}$ and $B = \{10, 4, 9\}$.

Then $A \cap B = \{4\}$.

If $A \cap B = \emptyset$, then A and B are called **disjoint**.

Properties of intersection and union

- $A \cap \emptyset = \emptyset$ — **domination law**
- $A \cap B = B \cap A$ — **commutative law**
- $(A \cap B) \cap C = A \cap (B \cap C)$ — **associative law**

  So we can meaningfully write, say, $A \cap B \cap C \cap D \cap E$ or $\bigcap_{i=1}^{n} A_i$.
- $A \cap A = A$ — **idempotent law**
- $A \subseteq B$ iff $A \cap B = A$
- **distributive laws:** $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

  $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
- BUT WATCH OUT! often $A \cap (B \cup C) \neq (A \cap B) \cup C$

  So writing $A \cap B \cup C$ might NOT make sense!

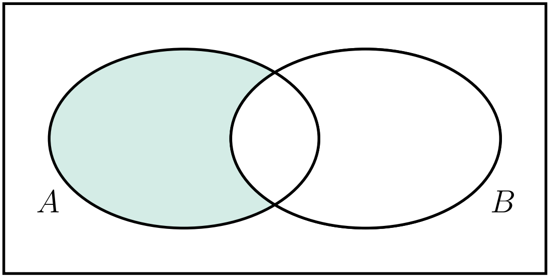
Set operations: difference

The **difference** of sets A and B is the set

$$\boxed{A - B \;=\; \{x \mid x \in A \text{ and } x \notin B\}}\,.$$

$A - B$ consists of those elements that are in A but not in B.

$A - B$ is also called the **complement of B with respect to A**.



Venn diagram of $A - B$

FOR EXAMPLE: $A = \{4, 7, 8\}$ and $B = \{10, 4, 9\}$.

Then $A - B = \{7, 8\}$ and $B - A = \{10, 9\}$.

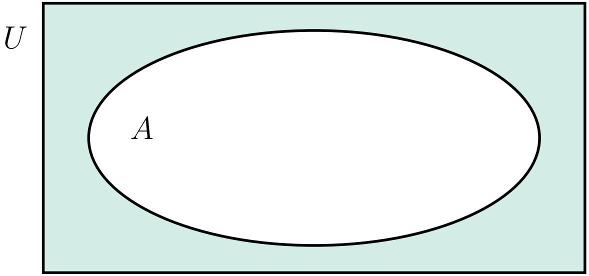
Set operations: (absolute) complement

In certain contexts we may consider all sets under consideration as being subsets of some given **universal set** U

(it is like the domain of a quantification).

Given a universal set U and $A \subseteq U$, the **complement of A** (w.r.t. U) is the set

$$\boxed{\bar{A} \;=\; U - A \;=\; \{x \in U \mid x \notin A\}}\,.$$



Venn diagram of $\bar{A}$

More properties of set operations

- **complementation laws:** $A \cup \bar{A} = U$

  $A \cap \bar{A} = \emptyset$

  $\overline{\bar{A}} = A$

  $\bar{U} = \emptyset$

  $\bar{\emptyset} = U$

- **De Morgan's laws:** $\overline{A \cup B} = \bar{A} \cap \bar{B}$

  $\overline{A \cap B} = \bar{A} \cup \bar{B}$

- $A - B = A \cap \bar{B}$

- $A \subseteq B$ iff $\bar{B} \subseteq \bar{A}$

Exercise 2.1 (examples of general direct proofs)

Prove that, for **any** X, Y and Z, we **always** have $X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$.

SOLUTION: According to lecture slide 60, we need to show both

(1) $X \cap (Y \cup Z) \subseteq (X \cap Y) \cup (X \cap Z)$ and **(2)** $(X \cap Y) \cup (X \cap Z) \subseteq X \cap (Y \cup Z)$.

Watch out! **It is not enough to give (or draw) some example sets X, Y, Z for which (1) and (2) hold.**

We need to show (1) and (2) using **no** assumptions on the sets X, Y, Z and their elements, using **only** the properties of set operations $\cup$ and $\cap$.

For (1): We need to show that every element of $X \cap (Y \cup Z)$ is also an element of $(X \cap Y) \cup (X \cap Z)$. Take an arbitrary ☺ $\in X \cap (Y \cup Z)$. Then both ☺ $\in X$ and ☺ $\in Y \cup Z$. There are two cases: either ☺ $\in Y$ or ☺ $\in Z$. In the first case, ☺ $\in X \cap Y$. In the second case ☺ $\in X \cap Z$. So in either case, ☺ $\in (X \cap Y) \cup (X \cap Z)$.

For (2): Take an arbitrary ☹ $\in (X \cap Y) \cup (X \cap Z)$. Then there are two cases: either ☹ $\in X \cap Y$, or ☹ $\in X \cap Z$. In the first case, both ☹ $\in X$ and ☹ $\in Y$. In the second case, both ☹ $\in X$ and ☹ $\in Z$. So ☹ $\in X$ in both cases, and either ☹ $\in Y$ or ☹ $\in Z$, depending on the case. Therefore, ☹ $\in Y \cup Z$, and so ☹ $\in X \cap (Y \cup Z)$.

Exercise 2.2

Show that there are sets A and B such that $A \cup \bar{B} \neq \bar{A} \cup B$.

SOLUTION: We need to find and describe **some** sets A and B such that $A \cup \bar{B}$ and $\bar{A} \cup B$ are different sets. There can be many possible solutions, here is one:

Let the universal set be $U = \{1, 2, 3\}$. Let $A = \{1\}$ and $B = \{2\}$.

Then $\bar{A} = \{2, 3\}$ and $\bar{B} = \{1, 3\}$. So $A \cup \bar{B} = \{1, 3\}$ and $\bar{A} \cup B = \{2, 3\}$.

These are two different sets as, for example, $1 \in A \cup \bar{B}$, but $1 \notin \bar{A} \cup B$.

Sequences

A **sequence** is a list of things, taken in a certain order.

To distinguish sequences from sets, we use brackets (instead of braces) when we list the elements of a sequence:

$(1, 5, 8, -2, 3)$ (*Cinderella*, *Tasmania*, *Tuesday*)

Important features of sequences:

- **The order of listing DOES matter:**
  $(1, 2, 3)$, $(1, 3, 2)$ and $(3, 1, 2)$ are different sequences.

- **Repeated occurrences DO matter:**
  (H, E, L, L, O), (H, H, H, E, L, L, O) and (H, E, L, O) are different sequences.

Tuples

Finite sequences are called **tuples**. A sequence with k elements is a **k-tuple**. The number k is called the **length** of the tuple.

Two tuples are **equal** if they have the same length and their corresponding elements are the same:

$\boxed{(x_1, x_2, \ldots, x_k) = (y_1, y_2, \ldots, y_n)}$ means that $k = n$ and

$x_1 = y_1$ and $x_2 = y_2$ $\ldots$ and $x_k = y_k$.

A 2-tuple is also called an **ordered pair**.

$\boxed{(a, b) = (c, d)}$ means that $a = c$ and $b = d$.

Cartesian product of sets

The **Cartesian product of sets A and B** is the set

$$\boxed{A\times B \;=\; \{(x,y) \mid x\in A \text{ and } y\in B\}}.$$

$A\times B$ consists of those ordered pairs (x,y) where $x\in A$ and $y\in B$.

FOR EXAMPLE: Let $A=\{1,2\}$ and $B=\{a,b,c\}$.
Then $A\times B \;=\; \{(1,a),(2,a),(1,b),(2,b),(1,c),(2,c)\}$.

The **Cartesian product of sets A_1, A_2, . . . , A_k** is the set

$$\boxed{A_1\times A_2\times\cdots\times A_k \;=\; \{(x_1,x_2,\dots,x_k) \mid x_i\in A_i \text{ for } i=1,2,\dots,k\}}.$$

$A_1\times A_2\times\cdots\times A_k$ consists of those k-tuples $(x_1,x_2,\dots,x_k)$
where $x_1\in A_1$, $x_2\in A_2$, . . . , $x_k\in A_k$.

FOR EXAMPLE: Let $A=\{1,2\}$, $B=\{a,b,c\}$ and $C=\{\Diamond,\Box\}$.

Then $A\times B\times C \;=\; \{(1,a,\Diamond),(1,a,\Box),(2,a,\Diamond),(2,a,\Box),(1,b,\Diamond),(1,b,\Box),(2,b,\Diamond),$
$(2,b,\Box),(1,c,\Diamond),(1,c,\Box),(2,c,\Diamond),(2,c,\Box)\}$.

Binary relations

For sets A and B, a **(binary) relation from A to B** is *any* subset R of the Cartesian product $A \times B$. (So a binary relation is a set consisting of some ordered pairs.)

We use notation $\boxed{aRb}$ to denote that $(a,b) \in R$, and say that **a is R-related to b**. If $(a,b) \notin R$, then we write $\boxed{a \not R\, b}$.

(Other notations: $R(a,b)$ for aRb, $\neg R(a,b)$ for $a \not R\, b$)

FOR EXAMPLE:

Let P be the set of people, and C the set of football clubs in the English Premier League. Define a relation Fan_of by taking

$$\text{Fan\_of} = \{(x,y) \in P \times C \mid x \text{ is a fan of } y\}.$$

If I am a fan of MU, but can't stand Arsenal, then

(*Agi*, *MU*) ∈ Fan_of   but   (*Agi*, *Arsenal*) ∉ Fan_of

We can also write the same things as

Agi Fan_of *MU*   and   *Agi* F$\not$an_of *Arsenal*

Relations on a set

A relation from a set A to A itself is called a **relation on A**.

In other words: a relation on a set A is a subset of $A \times A$

a relation on a set A is a set consisting *some* ordered pairs of elements from A

FOR EXAMPLE:

Let A be the set of all people in the class.

$$R_1 = \{(u, v) \in A \times A \mid u \text{ likes } v\}$$

$$R_2 = \{(u, v) \in A \times A \mid u \text{ is taller than } v\}$$

Raj is 180 cm tall and likes Jill who is 165 cm. Unfortunately, Jill doesn't like Raj.

Joe is 190 cm tall, and they don't like each other with Jill.

Then: (Raj , Jill) $\in R_1$ (Raj , Jill) $\in R_2$ (Jill , Raj) $\notin R_1$ (Jill , Raj) $\notin R_2$

(Joe , Jill) $\notin R_1$ (Joe , Jill) $\in R_2$ (Jill , Joe) $\notin R_1$ (Jill , Joe) $\notin R_2$

Relations on a set: some more examples

Here are some relations on the set **Z** of integers:

- **'smaller than':** $\boxed{< \;\; = \;\; \{(x,y) \in \mathbf{Z} \times \mathbf{Z} \mid x \text{ is smaller than } y\}}$

  -1 is smaller than 0, so $(-1,0) \in\, <$ used more often: $-1 < 0$

  5 is smaller than 25, so $(5,25) \in\, <$ used more often: $5 < 25$

  -3 is smaller than -2, so $(-3,-2) \in\, <$ used more often: $-3 < -2$

  0 is not smaller than -1, so $(0,-1) \notin\, <$ used more often: $0 \not< -1$

  10 is not smaller than 2, so $(10,2) \notin\, <$ used more often: $10 \not< 2$

  -4 is not smaller than -4, so $(-4,-4) \notin\, <$ used more often: $-4 \not< -4$

- **'smaller than or equal to':** $\boxed{\leq \;\; = \;\; \{(x,y) \in \mathbf{Z} \times \mathbf{Z} \mid x \text{ is smaller than or equal to } y\}}$

  $-1 \leq -1$, $5 \leq 6$, but $1 \not\leq 0$, $-2 \not\leq -10$

  $\boxed{< \subset \leq}$

- **'divisibility':** $\boxed{\text{div} \;\; = \;\; \{(x,y) \in \mathbf{Z} \times \mathbf{Z} \mid x \text{ divides } y\}}$

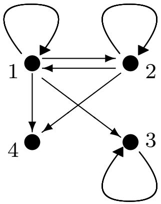
  $(1,25) \in \text{div}$, $(-3,12) \in \text{div}$, $(10,1000) \in \text{div}$, but $(0,4) \notin \text{div}$, $(2,-21) \notin \text{div}$, $(6,3) \notin \text{div}$

Representing relations

FOR EXAMPLE: Let $A = \{1, 2, 3, 4\}$ and take the following relation R on A:

$$R = \{(1,1),(1,2),(1,3),(1,4),(2,1),(2,2),(2,4),(3,3)\}.$$

Then R can be represented by 'points and arrows':



directed graph

Or, by a 0-1 **matrix:**

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 0 & 0 & 1 & 0 \\ 4 & 0 & 0 & 0 & 0 \end{array} \qquad \begin{array}{c|cccc} & 2 & 4 & 1 & 3 \\ \hline 2 & 1 & 1 & 1 & 0 \\ 4 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 3 & 0 & 0 & 0 & 1 \end{array} \qquad \begin{array}{c|cccc} & 4 & 3 & 1 & 2 \\ \hline 4 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 & 1 \end{array}$$

we can choose **any** order of listing of the set A, but we must choose the **same** order **both** from left-to-right and from top-to-bottom

Properties of relations

- A relation R on a set A is called **reflexive**,

  if $(a,a) \in R$ for every element $a \in A$.

  FOR EXAMPLE:

  - reflexive: $\leq$, $\geq$, $=$, ‘divisibility’ on $\mathbf{N}^+$

    $R_1 = \{(1,1),(1,2),(3,1),(2,2),(3,3)\}$ on $\{1,2,3\}$
  - not reflexive: $R_2 = \{(1,1),(1,2),(3,1),(3,3)\}$ on $\{1,2,3\}$

- A relation R on a set A is called **irreflexive**,

  if $(a,a) \notin R$ for every element $a \in A$.

  FOR EXAMPLE: $<$, $>$, $\neq$ on $\mathbf{N}$ or on $\mathbf{Z}$

  $R_3 = \{(1,2),(1,3),(2,1),(3,2)\}$ on $\{1,2,3\}$

Irreflexive is more than just ‘not reflexive’ !

Properties of relations (cont.)

- A relation R on a set A is called **symmetric**, if for all elements $a, b \in A$,

  $(b, a) \in R$ whenever $(a, b) \in R$.

  FOR EXAMPLE:

  - symmetric: $R_1 = \{(1,1),(1,2),(2,1),(2,3),(3,2),(3,3)\}$ on $\{1,2,3\}$

    $R_2 = \{(1,1),(2,2),(3,3)\}$ on $\{1,2,3\}$

    $\{(x, y) \in \mathbf{N} \times \mathbf{N} \mid x - y \text{ is divisible by } 3\}$ on $\mathbf{N}$

  - not symmetric: $R_3 = \{(1,1),(1,2),(2,1),(3,1),(3,3)\}$ on $\{1,2,3\}$

- A relation R on a set A is called **antisymmetric**, if

  (a, b) and (b, a) cannot be both in R unless $a = b$.

  (It does NOT mean that (a, a) should be in R !)

  FOR EXAMPLE: $<$, $>$, $\leq$, $\geq$ on **N**, **Z**, **Q**, or **R**

  $R_4 = \{(1,1),(1,3),(2,3),(3,3)\}$ on $\{1,2,3\}$

  R_2 above

Symmetric and antisymmetric are not opposites !

Properties of relations (cont.)

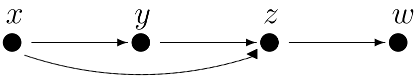
A relation R on a set A is called **transitive**, if the following hold, **for all elements** $a, b, c \in A$:

if **both** $(a,b) \in R$ **and** $(b,c) \in R$, **then** $(a,c) \in R$.

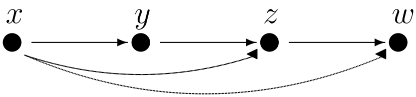
In the directed graph representing R :

R is transitive, if **every** two-step journey along arrows can be done in one step.

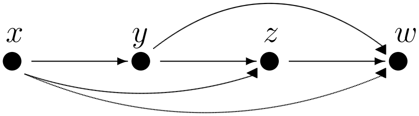
FOR EXAMPLE:



not transitive



still not transitive



transitive

Transitive relations: more examples and non-examples

- Transitive:

  $<$, $>$, $\leq$, $\geq$ on **N**, **Z**, **Q**, or **R**

  $R_1 = \{(1,1),(1,2),(1,3),(1,4),(2,2),(2,3),(2,4),(3,3),(3,4),(4,4)\}$

  $R_2 = \{(3,4)\}$ (both R_1 and R_2 on $\{1,2,3,4\}$)

- Not transitive:

  $R_3 = \{(1,1),(1,2),(2,1),(2,2),(3,4),(4,1),(4,4)\}$

  $R_4 = \{(1,1),(1,2),(2,1)\}$

  $R_5 = \{(1,1),(1,2),(1,4),(2,1),(2,2),(3,3),(4,1),(4,4)\}$

  (all R_3, R_4, R_5 on $\{1,2,3,4\}$)

Transitive closure

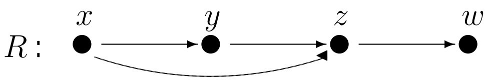
- Let R be a relation on a set A.

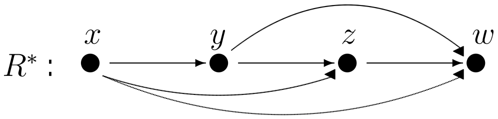
  The **transitive closure of** R is the smallest transitive relation on A containing R.

- We denote the transitive closure of R by $\boxed{R^*}$.

- (If R is already transitive, then $R^* = R$.)

  FOR EXAMPLE:





- Given R, we can define R^* recursively:

  Basis step: $R \subseteq R^*$

  (the pairs in R are all in R^*)

  Recursive step: If $(a, b) \in R^*$ and $(b, c) \in R^*$ then $(a, c) \in R^*$.

  (we add the missing pairs step-by-step)

Computing the transitive closure: Warshall's algorithm

An **algorithm** is a finite sequence of precise step-by-step instructions.

Warshall's algorithm computes the matrix of the transitive closure R^* of R.

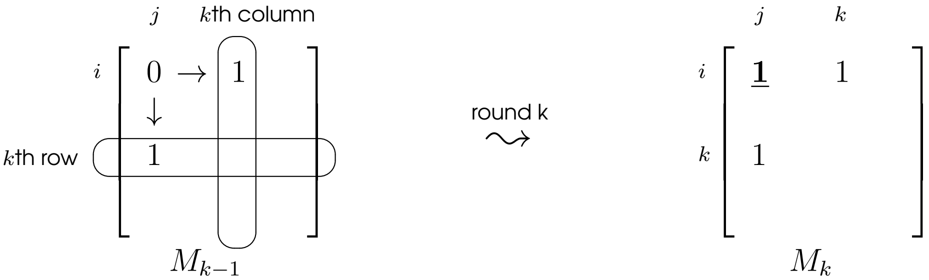
- Given a relation R on a set A with n elements, we begin with its $n \times n$ matrix $\boxed{M_0}$.
  (**any** order of listing A is fine)
- There are n rounds.

  In each round, we turn the previous matrix to a new matrix:

  $$M_0 \overset{\text{round 1}}{\leadsto} M_1 \overset{\text{round 2}}{\leadsto} M_2 \overset{\text{round 3}}{\leadsto} \dots \overset{\text{round n}}{\leadsto} M_n$$

  $\boxed{M_n}$ is the matrix of the transitive closure R^* of R.

  - 1st rule. we never change a 1 to 0
  - 2nd rule. rule for changing **some** 0 s to 1 s:



Warshall's algorithm: an example

Let R be a relation on $\{a, b, c, d\}$: $\boxed{R = \{(a, d), (b, a), (b, c), (c, a), (c, d), (d, c)\}}$

As $\{a, b, c, d\}$ has 4 elements, $n = 4$. The matrix of R is the 4×4 matrix

$$M_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

(here the chosen order of listing is a, b, c, d)

There will be 4 rounds.

Round 1.

$$\begin{array}{cc} & \text{1st column} \\ \text{1st row} & \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & \underline{0} \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\ & M_0 \end{array} \quad \overset{\text{round 1}}{\leadsto} \quad \begin{array}{c} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & \underline{\mathbf{1}} \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\ M_1 \end{array}$$

Warshall's algorithm: an example (cont.)

Round 2.

$$M_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \overset{\text{round 2}}{\rightsquigarrow} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = M_2 \quad \text{(no change)}$$

Round 3.

$$M_2 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ \underline{0} & 0 & 1 & \underline{0} \end{bmatrix} \overset{\text{round 3}}{\rightsquigarrow} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ \underline{\mathbf{1}} & 0 & 1 & \underline{\mathbf{1}} \end{bmatrix} = M_3$$

Round 4.

$$M_3 = \begin{bmatrix} \underline{0} & 0 & \underline{0} & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & \underline{0} & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \overset{\text{round 4}}{\rightsquigarrow} \begin{bmatrix} \underline{\mathbf{1}} & 0 & \underline{\mathbf{1}} & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & \underline{\mathbf{1}} & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} = M_4$$

$$\boxed{R^* = \{(a,a),(a,c),(a,d),(b,a),(b,c),(b,d),(c,a),(c,c),(c,d),(d,a),(d,c),(d,d)\}}$$

Equivalence relations

A relation R on a set A is called an **equivalence relation** if it is

- reflexive,
- symmetric, and
- transitive.

FOR EXAMPLE:

- $=$ on any set,
- $\{(x,y) \in \mathbf{Z} \times \mathbf{Z} \mid x - y$ is divisible by $4\}$ on $\mathbf{Z}$

Partial orders

A relation R on a set A is called a **partial order** if it is

- reflexive,
- antisymmetric, and
- transitive.

FOR EXAMPLE: $\leq$, $\geq$, and ‘divisibility’ on $\mathbf{N}^+$

EXERCISE: Prove that $\subseteq$ is a partial order on the power set $P(S)$ of a set S.

SOLUTION:

- $\subseteq$ is reflexive: It is because $A \subseteq A$ for every set A.
- $\subseteq$ is antisymmetric: If $A \subseteq B$ and $B \subseteq A$, then $A = B$ holds.
- $\subseteq$ is transitive: If $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$ holds.

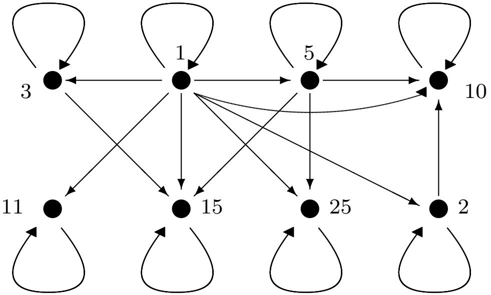
Representing partial orders: Hasse diagrams

If we **know that a relation is a partial order**, then there is a more `economical' way of representing it than by a directed graph.

Say, take the `divisibility' relation on the set $\{1, 2, 3, 5, 10, 11, 15, 25\}$:

$$\{(1,1),(1,2),(1,3),(1,5),(1,10),(1,11),(1,15),(1,25),(2,2),(2,10),(3,3),\\(3,15),(5,5),(5,10),(5,15),(5,25),(10,10),(11,11),(15,15),(25,25)\}$$

As this is a relation, it can be represented by a directed graph:

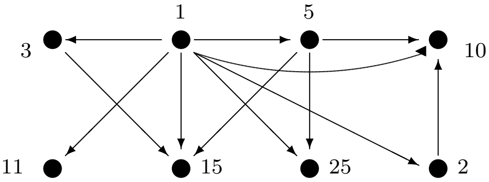


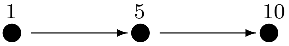
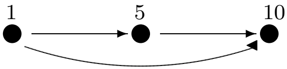
But we can do better, **using** our knowledge while drawing.

Constructing Hasse diagrams (cont.)

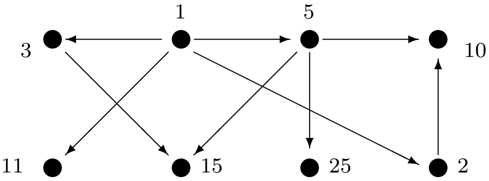
As partial orders are always **reflexive,** a loop is always present at every point.

So by removing these loops we don't lose info:



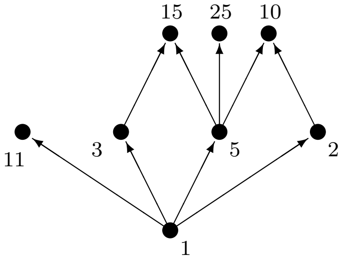
Partial orders are always **transitive.** Say, if  is part of our diagram, then we know that we must also have 

So we don't lose info by indicating only 'one-step' arrows, and removing the rest:

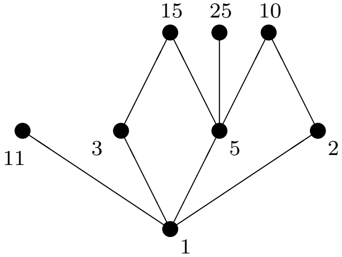


Constructing Hasse diagrams (cont.)

Partial orders are always **antisymmetric.** This means that between any two points there can be an arrow one way only, NOT both. So we can rearrange the points such that all the arrows 'point' from a lower position 'upwards':



So we don't lose info by removing the arrow-heads, and using lines instead:

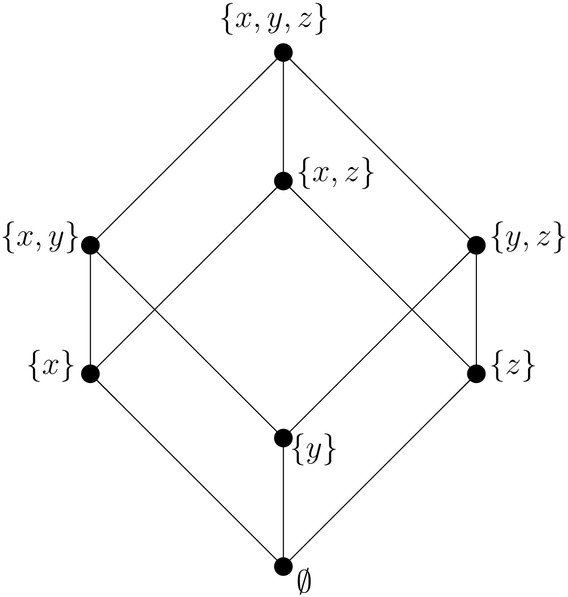


Hasse diagram
of the 'divisibility' relation
on $\{1, 2, 3, 5, 10, 11, 15, 25\}$

(Overall shape does not matter, but WATCH OUT: 'horizontal' lines are NO GOOD!)

Hasse diagrams: another example

The Hasse diagram of $\subseteq$ on the power set $P(\{x,y,z\})$ of $\{x,y,z\}$:



Linear orders

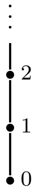
A relation R on a set A is called a **linear order** (or **total order**) if

- R is a partial order, and
- for all $a, b \in A$, either $(a, b) \in R$ or $(b, a) \in R$ (that is, every pair of elements from A is in R this way or the other).

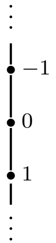
The name 'linear' comes from the fact that the Hasse diagram of a linear order is always a **line.**

FOR EXAMPLE:

- $\leq$ on **N**:



- $\geq$ on **Z**:



- BUT: 'divisibility' and $\subseteq$ are partial orders, but NOT linear orders (see previous two lecture slides)

Exercise 2.3

Let S be a set with more than one element.

Show that $\subseteq$ is **not** a linear order on $P(S)$.

SOLUTION:

If S has more than one element, then there are at least two different elements in S, let's call them x and y. Then:

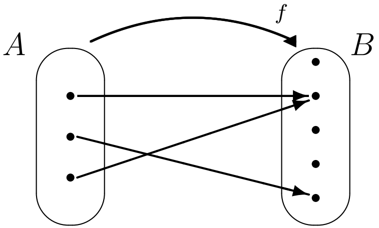
- $x \in S$, so $\{x\} \subseteq S$, and so $\{x\} \in P(S)$.
- $y \in S$, so $\{y\} \subseteq S$, and so $\{y\} \in P(S)$.
- Neither $\{x\} \subseteq \{y\}$ nor $\{y\} \subseteq \{x\}$ holds, as x and y are different.

So $\{x\}$ and $\{y\}$ are two elements in $P(S)$ such that neither $\{x\} \subseteq \{y\}$ nor $\{y\} \subseteq \{x\}$, and so $\subseteq$ is not a linear order.

(Check the Hasse diagram on lecture slide 91. It is not a line.)

Functions

Given sets A and B, a **function from A to B** is a rule f that associates with *each* element of A *exactly one* element of B.



If f associates $x \in A$ with $y \in B$, then we write $\boxed{f(x) = y}$ and say "f of x is y", or "f maps x to y", or the "value of f at x is y".

If f is function from A to B, then we write: $\boxed{f : A \to B}$.

We call A the **domain of f**, and B **the codomain of f**.

- **every** element of the domain has to be mapped somewhere in the codomain (but **not everything** in the codomain has to be a value of a domain element)
- one element **cannot** be mapped to 2 different places (but it **can** happen that 2 different elements are mapped to the same place)

Functions and not functions

Let H be the set of all humans, alive or dead. Let's make some $H \to H$ associations and discuss whether each is a $H \to H$ function or not.

- $\boxed{f(x) \text{ is a parent of } x}$

  This f is NOT a function, because people have two parents.

- $\boxed{f(x) \text{ is the mother of } x}$

  This f is a $H \to H$ function, because each person has exactly one mother.

- $\boxed{f(x) \text{ is the oldest child of } x}$

  This f is NOT a $H \to H$ function, because some person has no children.

- $\boxed{f(x) \text{ is the set of all children of } x}$

  Though this f is a function, it is NOT a $H \to H$ function, because each person is associated with a **set** of people rather than one person.
  (This f is a $H \to P(H)$ function.)

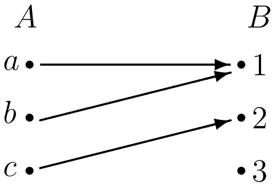
Different ways of describing functions

FOR EXAMPLE: Let $A = \{a, b, c\}$ and $B = \{1, 2, 3\}$.

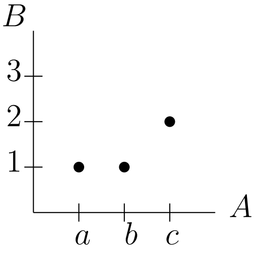
- We can describe a function $f : A \to B$ by listing all its associations:

$$f(a) = 1, \qquad f(b) = 1, \qquad f(c) = 2\,.$$

- We can describe the same f by drawing points and arrows:



- We can describe the same f by drawing its '**graph**':



More examples of $f : A \to B$ functions

- A function is a rule. Sometimes we can describe this rule by a single formula:

  Let $A = B = \mathbf{Z}$. For every $x \in \mathbf{Z}$, let

$$\boxed{f(x) \; = \; 6x + 28}$$

  Then, for example, $f(2) = 6 \cdot 2 + 28 = 40$, $f(0) = 28$, $f(-113) = -650$, $\dots$

- Sometimes the rule can only be described by case distinction:

  Let $A = B = \mathbf{N}$. For every $n \in \mathbf{N}$, let

$$\boxed{g(n) \; = \; \begin{cases} 2^n, & \text{if } n \text{ is odd,} \\ 3n^2 + n + 1, & \text{if } n \text{ is even} \end{cases}}$$

  Then, for example, $g(4) = 3 \cdot 4^2 + 4 + 1 = 53$, $g(3) = 2^3 = 8$, $\dots$

- And the rule does not have to be described by a formula at all:

  Let $A = \{E \mid E \text{ is a healthy African elephant}\}$ and

  $B = \{e \mid e \text{ is an elephant ear}\}$.

  For every $E \in A$, let $\boxed{\ell(E) \; = \; E\text{'s left ear}}$

Some useful functions

- The **floor function** $\boxed{\lfloor\ \rfloor : \mathbf{R} \to \mathbf{Z}}$ assigns to any real number x the largest integer that is less than or equal to x.

  FOR EXAMPLE: $\lfloor \frac{1}{2} \rfloor = 0, \quad \lfloor -\frac{3}{2} \rfloor = -2, \quad \lfloor 3.2 \rfloor = 3, \quad \lfloor 9 \rfloor = 9.$

- The **ceiling function** $\boxed{\lceil\ \rceil : \mathbf{R} \to \mathbf{Z}}$ assigns to any real number x the smallest integer that is greater than or equal to x.

  FOR EXAMPLE: $\lceil \frac{1}{3} \rceil = 1, \quad \lceil -\frac{5}{4} \rceil = -1, \quad \lceil 5.3 \rceil = 6, \quad \lceil 7 \rceil = 7.$

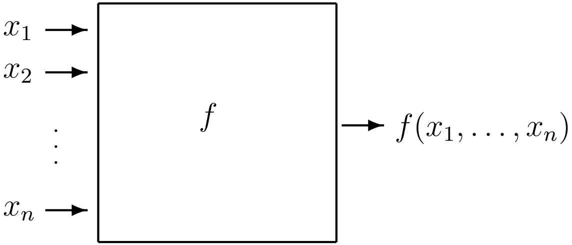
$$\boxed{\begin{gathered} x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1 \\ \lfloor -x \rfloor = -\lceil x \rceil \\ \lceil -x \rceil = -\lfloor x \rfloor \end{gathered}}$$

Functions with multiple arguments

If the domain of a function f is a Cartesian product $A_1 \times \cdots \times A_n$, we say that **$f$ has arity n**, or **f is an n-ary function**, or **f has n arguments**.

In this case, for each n-tuple $(x_1, \dots, x_n) \in A_1 \times \cdots \times A_n$,

$\boxed{f(x_1, \dots, x_n)}$ denotes the value of f at $(x_1, \dots, x_n)$.



A function f with two arguments is also called a **binary function.**

For binary functions, we have the option of writing $f(x, y) = z$ in the form $\boxed{x \, f \, y = z}$ (such as, $4 + 5 = 9$ instead of $+(4, 5) = 9$).

Tuples and sequences are functions

- A tuple can be thought of as a function.

  FOR EXAMPLE: The 5-tuple $(22, 14, 55, 1, 700)$ can be thought of as a listing of the values of the function $f : \{0, 1, 2, 3, 4\} \to \mathbf{N}$ defined by

  $$f(0) = 22, \qquad f(1) = 14, \qquad f(2) = 55, \qquad f(3) = 1, \qquad f(4) = 700.$$

- Similarly, an infinite sequence of objects can also be thought of as a function.

  FOR EXAMPLE: Suppose that $(b_0, b_1, \ldots, b_n, \ldots)$ is an infinite sequence of objects from a set S. Then this sequence can be thought of as a listing of the values of the function $f : \mathbf{N} \to S$ defined by

  $$f(n) = b_n.$$

Functions are special binary relations

A function $f : A \to B$ can be considered as a relation from A to B:

$$\boxed{\{(a,b) \in A \times B \mid f(a) = b\}}$$

Relations that are also functions have two special properties:

- for every $a \in A$ there is some $b \in B$ with (a,b) being in the relation (**every** element of the domain has to be mapped somewhere in the codomain)
- and no two ordered pairs in the relation have the same first element (one element **cannot** be mapped to 2 different places)

FOR EXAMPLE:	WHY?
$\{(x,y) \in \mathbf{N} \times \mathbf{N} \mid y = x - 1\}$ is NOT a function.	0 is not mapped
$\{(x,y) \in \mathbf{Z} \times \mathbf{Z} \mid y = x - 1\}$ is a function.	$f(n) = n - 1$
$\{(x,y) \in \mathbf{N} \times \mathbf{R} \mid x = y^2\}$ is NOT a function.	$(25, 5)$ and $(25, -5)$ are both in
$\{(x,y) \in \mathbf{N} \times \mathbf{N} \mid x = y^2\}$ is NOT a function.	2 is not mapped
$\{(x,y) \in \mathbf{N} \times \mathbf{N} \mid y = x^2\}$ is a function.	$f(n) = n^2$

Properties of functions

- A function $f : A \to B$ is called **one-to-one** (or **injective**)

  if it maps distinct elements of A to distinct elements of B.

  Another way to say this: f is one-to-one if

  for all elements x, y in A, if $x \neq y$ then $f(x) \neq f(y)$.

- A function $f : A \to B$ is called **onto** (or **surjective**)

  if every element b in B can be obtained as $b = f(a)$ for some a in A.

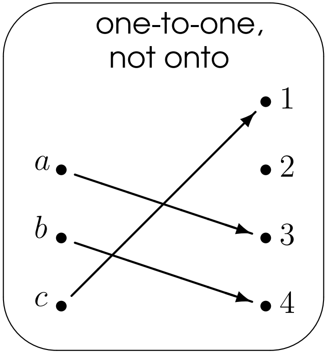
  In general, this might not be the case. If f is an $A \to B$ function, this just means that for every $a \in A$, we have $f(a) \in B$. But the **range of f**

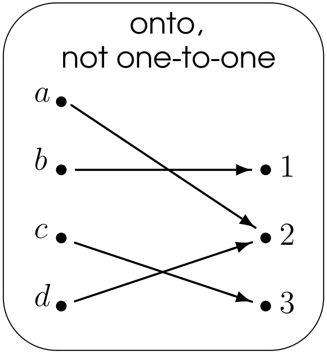
  $$\boxed{\text{range}(f) \;=\; \{f(a) \in B \mid a \in A\}}$$

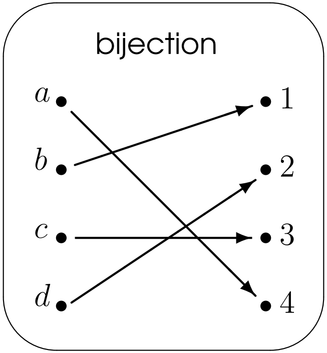
  can be a **proper subset** of B.

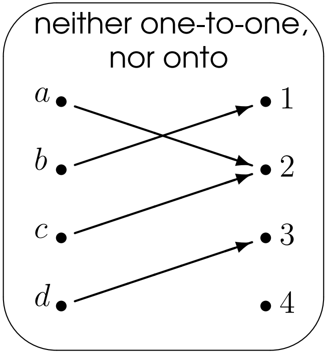
- A function is called a **bijection** if it is both one-to-one and onto.

Examples


one-to-one,
not onto

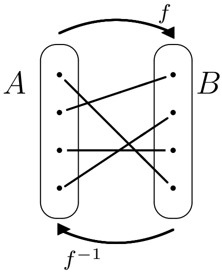

onto,
not one-to-one


bijection


neither one-to-one,
nor onto

Bijections and inverses

Bijections always come in pairs. If $f : A \to B$ is a bijection, then there is a function $\boxed{f^{-1} : B \to A}$, called the **inverse of f** defined by



$$\boxed{f^{-1}(b) = a \qquad \text{whenever} \qquad f(a) = b}$$

Then f^{-1} is also a bijection, and we have

- $f^{-1}\big(f(a)\big) = a$, for every $a \in A$, $\qquad f\big(f^{-1}(b)\big) = b$, for every $b \in B$.

FOR EXAMPLE: Let *Odd* and *Even* be the sets of odd and even natural numbers, respectively. Define a function $f : \mathit{Odd} \to \mathit{Even}$ by

$$f(n) = n - 1 \,.$$

Then f is a bijection and its inverse $f^{-1} : \mathit{Even} \to \mathit{Odd}$ is defined by

$$f^{-1}(n) = n + 1 \,.$$

Some important functions

- For every set A, its **identity function** $\mathrm{id}_A : A \to A$ is defined by, for all $a \in A$,

  $$\boxed{\mathrm{id}_A(a) = a}$$

  Then id_A is a bijection and its inverse is itself.

- Let S be a set. For every subset $A \subseteq S$, its **characteristic function** $f_A : S \to \{0,1\}$ is defined by, for all $x \in S$,

  $$\boxed{f_A(x) = \begin{cases} 1, & \text{if } x \in A, \\ 0, & \text{if } x \notin A \end{cases}}$$

  Say, if $A \subseteq \mathbf{N}$ then $f_A : \mathbf{N} \to \{0,1\}$ can be represented by an infinite 0-1 sequence.

Describing N → N functions by recursion

- The **factorial function**:

  Basis step: $f(0) = 1$ and $f(1) = 1$.

  Recursive step: If $n > 1$ then $f(n) = f(n-1) \cdot n$.

  We used to write $\boxed{n!}$ for $f(n)$.

- The **Fibonacci function**:

  Leonardo Fibonacci asked in 1202: Let's start with a pair of rabbits that needs one month to mature, and assume that every month each pair produces a new pair that becomes productive after one month. How many **new pairs** are produced each month?

  Basis step: $f(0) = 0$ and $f(1) = 1$.

  Recursive step: If $n > 1$ then $f(n) = f(n-2) + f(n-1)$.

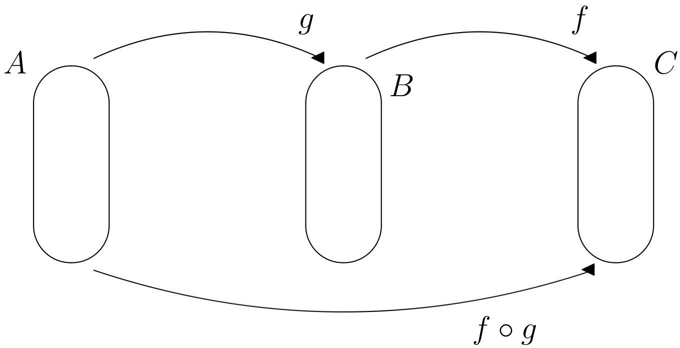
  $0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ \ldots$

Combining functions: composition

Let $g : A \to B$ and $f : B \to C$ be functions.

The **composition of f and g** is the function $\boxed{(f \circ g) : A \to C}$ defined by

$$\boxed{(f \circ g)(a) \;=\; f\big(g(a)\big)} \qquad \text{for each } a \in A.$$



We **only** define the composition $f \circ g$ when

"codomain of g" $=$ "domain of f" !

Composition of functions: examples

Let $X = \{a, b, c\}$ and $Y = \{1, 2, 3\}$. Let function $g : X \to X$ be defined by

$$g(a) = b, \qquad g(b) = c, \qquad g(c) = a,$$

and function $f : X \to Y$ be defined by

$$f(a) = 3, \qquad f(b) = 2, \qquad f(c) = 1.$$

Then:

- $(f \circ g)(a) = f\big(g(a)\big) = f(b) = 2$,

  $(f \circ g)(b) = f\big(g(b)\big) = f(c) = 1$,

  $(f \circ g)(c) = f\big(g(c)\big) = f(a) = 3$.

- $(g \circ g)(a) = g\big(g(a)\big) = g(b) = c$,

  $(g \circ g)(b) = g\big(g(b)\big) = g(c) = a$,

  $(g \circ g)(c) = g\big(g(c)\big) = g(a) = b$.

- Watch out: $f \circ f$ and $g \circ f$ are not defined!

Properties of composition

- Even if both are defined, $f \circ g$ and $g \circ f$ can be different: Let f and g be both **Z** → **Z** functions, defined by $f(x) = 2x + 3$ and $g(x) = 3x + 2$. Then:

$$\begin{aligned} (f \circ g)(x) &= f\big(g(x)\big) = f(3x+2) = 2(3x+2)+3 = 6x+7, \\ (g \circ f)(x) &= g\big(f(x)\big) = g(2x+3) = 3(2x+3)+2 = 6x+11. \end{aligned}$$

- $\circ$ is associative: $\boxed{f \circ (g \circ h) = (f \circ g) \circ h}$
- If $f : A \to B$ is a bijection then

$$\boxed{f^{-1} \circ f = \mathsf{id}_A \qquad \text{and} \qquad f \circ f^{-1} = \mathsf{id}_B}$$

- For any function $f : A \to B$,

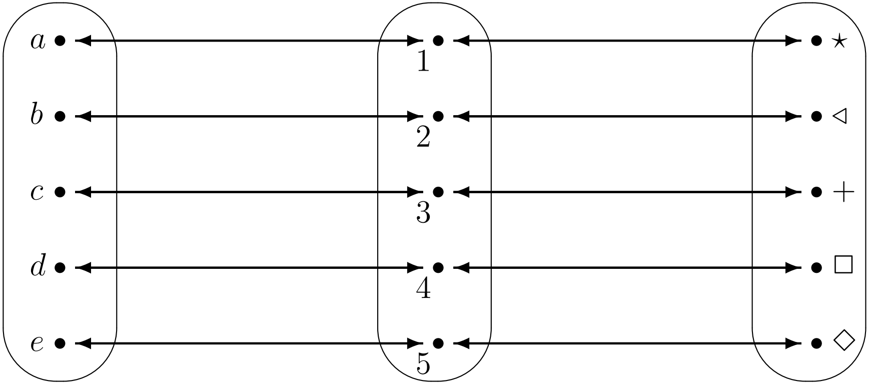
$$\boxed{f \circ \mathsf{id}_A = \mathsf{id}_B \circ f = f}$$

Comparing finite sets

It is not hard to compare the sizes of finite sets:

we simply count the number of elements in each.

If finite sets have the same number of elements, then there is always a *bijection* between them.



And infinite sets?

Idea: Two infinite sets have the same size if there is bijection between them.

We call a set A **countable** if it is either finite

or there is a bijection between A and $\mathbf{N}$.

FOR EXAMPLE:

- As $\mathrm{id}_{\mathbf{N}} : \mathbf{N} \to \mathbf{N}$ is a bijection, $\mathbf{N}$ is countable.

- Let *Odd* be the set of odd natural numbers.

  Strangely enough, even if $\mathit{Odd} \subset \mathbf{N}$, it has the same size as $\mathbf{N}$:

  The function $f : \mathbf{N} \to \mathit{Odd}$ defined by

  $$f(x) = 2x + 1$$

  is a bijection.

$\mathbf{N} \times \mathbf{N}$ is countable

We need to describe a bijection between $\mathbf{N} \times \mathbf{N}$ and $\mathbf{N}$.

We arrange the ordered pairs in $\mathbf{N} \times \mathbf{N}$ in such a way that they can be easily counted:

$$\begin{array}{lcl}
(0,0) & \longleftrightarrow & 0, \\
(0,1),\ (1,0), & \longleftrightarrow & 1,\ 2, \\
(0,2),\ (1,1),\ (2,0), & \longleftrightarrow & 3,\ 4,\ 5, \\
(0,3),\ (1,2),\ (2,1),\ (3,0), & \longleftrightarrow & 6,\ 7,\ 8,\ 9, \\
\qquad \ldots & & \quad \ldots
\end{array}$$

We can describe this $\mathbf{N} \times \mathbf{N} \to \mathbf{N}$ bijection by

$$(m,n) \qquad \longleftrightarrow \qquad \big(1+2+\cdots+(m+n)\big)+m\,.$$

But not all sets are countable

The power set $P(\mathbf{N})$ of $\mathbf{N}$ is NOT countable:

$$\neg\exists f \, \big(f \text{ is a bijection between } \mathbf{N} \text{ and } P(\mathbf{N})\big)$$

In other words, *there are more subsets of numbers than numbers.*

WHY? We show that $\forall f \, \big(\text{if } f : \mathbf{N} \to P(\mathbf{N}) \text{ then } f \text{ is not onto}\big)$:

Let $f : \mathbf{N} \to P(\mathbf{N})$ be an arbitrary function.

Then, for every $n \in \mathbf{N}$, $f(n)$ is a subset of $\mathbf{N}$.

Now take the following subset D_f of $\mathbf{N}$: $\boxed{D_f \; = \; \{n \in \mathbf{N} \mid n \notin f(n)\}}$

We show that D_f **is not in the range of** f, that is, for every $n \in \mathbf{N}$, $\boxed{D_f \neq f(n)}$:

Take an arbitrary $n \in \mathbf{N}$. Then there are two cases, either $n \in D_f$ or $n \notin D_f$.

- If $n \in D_f$, then n should have the property describing D_f, so $n \notin f(n)$

  $\leadsto$ $\boxed{D_f \not\subseteq f(n)}$

- If $n \notin D_f$, then the property describing D_f does not hold for n, so $n \in f(n)$

  $\leadsto$ $\boxed{f(n) \not\subseteq D_f}$

Counting finite sets

We denote the size of a finite set S by

$$\boxed{\,|\,S\,|\,}$$

FOR EXAMPLE:

- If $S = \{a, b, c\}$ then $|S| = |\{a, b, c\}| = 3$.

  We say that "the cardinality of S is 3", or simply "S has 3 elements".

- If $A = \{n \in \mathbf{N} \mid n$ is an odd number between 4 and $24\}$ then $|A| = 10$.

- $|\emptyset| = 0$.

The sum rule

If A and B are *disjoint* sets then $|A \cup B| = |A| + |B|$.

If $A_1, A_2, \ldots, A_n$ are n *parwise disjoint* sets then

$$|A_1 \cup A_2 \cup \cdots \cup A_n| = |A_1| + |A_2| + \cdots + |A_n|.$$

FOR EXAMPLE:

A student can choose a project from one of the project-lists of three lecturers, John, Bill and Eve. John's list has 11 projects, Bill's 10, and Eve's 8. No project occurs in two lists. How many possible projects are there to choose from?

SOLUTION: $A = \{p \mid p$ is a project on John's list$\}$

$B = \{p \mid p$ is a project on Bill's list$\}$

$C = \{p \mid p$ is a project on Eve's list$\}$

Then the number of possible projects is

$$|A \cup B \cup C| = |A| + |B| + |C| = 11 + 10 + 8 = 29.$$

The inclusion-exclusion principle

$$\boxed{|A \cup B| \;=\; |A| + |B| - |A \cap B|}$$

FOR EXAMPLE: A company receives 350 applications from fresh graduates for a certain job. 220 of these people have studied computer science, 147 management, and 51 both computer science and management. How many of these applicants studied neither computer science nor management?

SOLUTION: Let $C \;=\; \{p \mid p$ is an applicant who studied CS$\}$,

$M \;=\; \{p \mid p$ is an applicant who studied management$\}$.

Then the number of those who studied at least one of the two is

$$|C \cup M| \;=\; |C| + |M| - |C \cap M| \;=\; 220 + 147 - 51 \;=\; 316\,.$$

So the number of those who studied neither computer science nor management is this number subtracted from the total number of applicants:

$$350 - 316 \;=\; 34\,.$$

Inclusion-exclusion principle for 3 sets

$$
\begin{aligned}
\boxed{|A \cup B \cup C|} &= |A \cup (B \cup C)| \\
&= |A| + |B \cup C| - |A \cap (B \cup C)| \\
&= |A| + |B| + |C| - |B \cap C| - |A \cap (B \cup C)| \\
&= |A| + |B| + |C| - |B \cap C| - |(A \cap B) \cup (A \cap C)| \\
&= |A| + |B| + |C| - |B \cap C| - \\
&\qquad - \big(|A \cap B| + |A \cap C| - |(A \cap B) \cap (A \cap C)|\big)
\end{aligned}
$$

$$= \boxed{|A| + |B| + |C| - |B \cap C| - |A \cap B| - |A \cap C| + |A \cap B \cap C|}$$

The product rule

If there is a sequence of k tasks such that

there are n_1 ways to do the first task,

n_2 ways to do the second task, $\dots$,

n_k ways to do the kth task,

then there are $n_1 \cdot n_2 \cdot \dots \cdot n_k$ ways to do the whole sequence of k tasks.

$$\boxed{|A_1 \times A_2 \times \cdots \times A_k| = |A_1| \cdot |A_2| \cdot \dots \cdot |A_k|}$$

FOR EXAMPLE:

If each number plate contains a sequence of three letters followed by three digits (and no such sequence is prohibited), then the number of available different number plates is:

$$26 \cdot 26 \cdot 26 \cdot 10 \cdot 10 \cdot 10 = 17\,576\,000$$

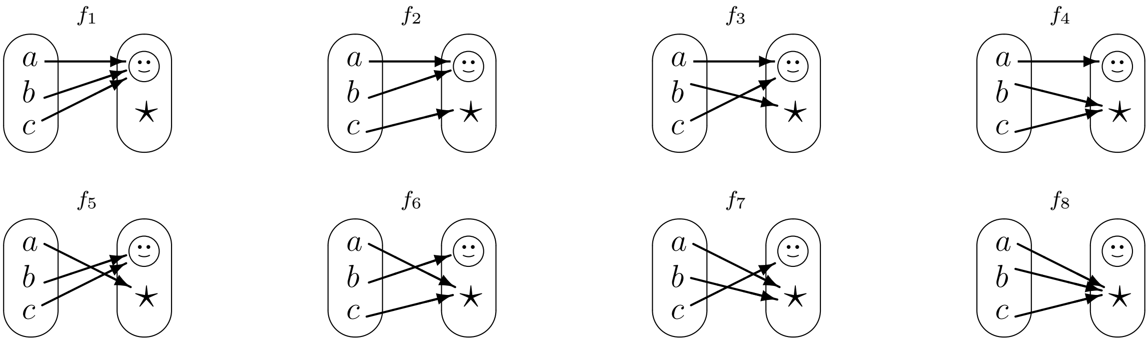
The product rule: another example

If $|A| = n$ and $|B| = m$ then the number of different $f : A \to B$ functions is:

$$\overbrace{m \cdot m \cdot \ \ldots\ \cdot m}^{n} = m^n$$

FOR EXAMPLE: $A = \{a, b, c\}$, $B = \{\text{☺}, \star\}$ $\rightsquigarrow$ $n = 3$, $m = 2$

- there are 2 ways to choose a value for a
- then there are 2 ways to choose a value for b
- then there are 2 ways to choose a value for c



Counting the subsets of a finite set

Let S be a finite set having n elements. How many subsets does S have? In other words, what is the cardinality of $P(S)$?

SOLUTION: List the elements of S: $s_1, s_2, \ldots, s_n$.

We can choose a subset A of S by going through this list and decide, for each element in the list, whether we choose it to be in A or not.

- So we have one 'task' for each element: n tasks altogether.
- Each task can be done in two ways: IN or OUT .

Therefore, by the product rule:

$$\boxed{|P(S)| = \overbrace{2 \cdot 2 \cdot \ \ldots \ \cdot 2}^{n} = 2^n}$$

Mixing the sum and product rules

How many four-digit numbers begin either with 7 or with 42?

SOLUTION:

- Our set can be broken into two disjoint subsets:

  (1) four-digit numbers beginning with 7, and

  (2) four-digit numbers beginning with 42.

  We deal with these separately, then apply the sum rule.

- In case (1), if we begin with 7, then there are 3 digits still to be filled in, with 10 choices each. So, by the product rule, we obtain: $10 \cdot 10 \cdot 10 = 1000$ possibilities.

- Similarly, in case (2), there are 2 digits still to be filled in, with 10 choices each. So we have $10 \cdot 10 = 100$ possibilities.

- By the sum rule, altogether we have: $1000 + 100 \; = \boxed{1100}$

The pigeonhole principle

If $k \in \mathbf{N}^+$ and $k+1$ or more objects are placed into k boxes, then there is at least one box containing two or more objects.

FOR EXAMPLE:

- Among any group of 367 people, there must be at least two with the same birthdays.
- If an exam is marked on a scale from 0 to 100, then among 102 students there must be at least two with the same mark.
- If A and B are finite sets with $|A| > |B|$ and $f : A \to B$ is a function, then f is not one-to-one.

  (*Hint:* Suppose that for each $b \in B$ we have a box that contains those $a \in A$ for which $f(a) = b$.)

The pigeonhole principle: another example

For every $n \in \mathbf{N}^+$ there is a multiple of n that is not 0, and has only 0s and 1s among its digits.

SOLUTION: Take any $n \in \mathbf{N}^+$. Consider the $n+1$ many numbers

$$1, \quad 11, \quad 111, \quad \ldots, \quad \overbrace{111\ldots1}^{n+1}$$

- Observe that there are n possible different remainders when a natural number is divided by n (these are: $0,\ 1,\ \ldots,$ or $n-1$).
- Because there are $n+1$ numbers on the above list, by the pigeonhole principle there must be two with the same remainder when divided by n.
- Then "<u>the larger of these numbers minus the smaller one</u>"
  - is divisible by n, and so is a multiple of n,
  - it has only 0s and 1s among its digits.

The generalised pigeonhole principle

If n objects are placed into k boxes, then
there is at least one box containing at least $\lceil n/k \rceil$ objects.

FOR EXAMPLE:

- Among 100 people there are at least $\lceil 100/12 \rceil = 9$ who where born in the same month.
- If the possible grades for a module are A, B, C, D, and F,

  then among 26 students there are always at least 6 with the same module grade (because $\lceil 26/5 \rceil = 6$).

Four ways of selecting items

A girl is given a bag containing three types of sweets:

aniseed drops (A), butter mints (B), and cherry drops (C).

In how many ways can she select two sweets?

It depends.

- If **order matters, repetition not allowed:**

  AB, AC, BA, BC, CA, CB

- If **order matters, repetition allowed:**

  AA, AB, AC, BA, BB, BC, CA, CB, CC

- If **order does not matter, repetition not allowed:**

  AB, AC, BC

- If **order does not matter, repetition allowed:**

  AA, AB, AC, BB, BC, CC

Order matters, repetition not allowed

How many ways can we select k persons from a group of n people to stand in line for a photo shoot?

There are:

- n ways to select the first person,
- $n-1$ ways to select the second person,
- $\ldots$ and so on, up to $n-(k-1)$ ways to select the kth person.

So, by the product rule, the overall number is:

$$\boxed{n\cdot(n-1)\cdot\ \ldots\ \cdot(n-k+1) = \frac{n\cdot(n-1)\cdot\ \ldots\ \cdot(n-k+1)\cdot(n-k)\cdot\ \ldots\ \cdot 2\cdot 1}{1\cdot 2\cdot\ \ldots\ \cdot(n-k)} = \frac{n\,!}{(n-k)\,!}}$$

SPECIAL CASE: $k=n$, the number of ways k objects can be ordered:

$$\frac{k\,!}{(k-k)\,!} = \frac{k\,!}{0\,!} = \frac{k\,!}{1} = \boxed{k\,!} = 1\cdot 2\cdot\ \ldots\ \cdot k$$

Order does not matter, repetition not allowed

How many ways can we select k persons from a group of n people if the order of selection does not matter?

- As we have seen, if the order of selection does matter, then there are $n \cdot (n-1) \cdot \ldots \cdot (n-k+1) = \dfrac{n\,!}{(n-k)\,!}$ ways.
- But then for each set A of k people, we counted the people in A many times. How many? As many as the number of ways k persons can be ordered: $k\,! = 1 \cdot 2 \cdot \ldots \cdot k$
- So, if the selection order does not matter, then the number of ways is:

$$\frac{\frac{n\,!}{(n-k)\,!}}{k\,!} = \frac{n\,!}{(n-k)\,! \cdot k\,!} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{1 \cdot 2 \cdot \ldots \cdot k} = \boxed{\binom{n}{k}}$$

n **choose** k

(Other notations in the literature for the 'choose numbers': C^n_k, ${}_nC_k$, $C(n,k)$)

Properties of the ‘choose numbers’

- $$\binom{n}{k} = \frac{n \cdot (n-1) \cdot \ \ldots \ \cdot (n-k+1)}{1 \cdot 2 \cdot \ \ldots \ \cdot k} \qquad \frac{\leftarrow \ k \text{ `backward steps'}}{\leftarrow \ k \text{ `forward steps'}}$$

  FOR EXAMPLE:

  $$\binom{128}{3} = \frac{128 \cdot 127 \cdot 126}{1 \cdot 2 \cdot 3} \qquad \frac{\leftarrow \ 3 \text{ `backward steps'}}{\leftarrow \ 3 \text{ `forward steps'}}$$

- ‘choose numbers’ are also called **binomial coefficients**

- $$\binom{n}{k} = \binom{n}{n-k}$$

  WHY? choosing k elements is the same as ‘leaving’ $n-k$

Pascal's identity

$$\boxed{\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}}$$

$$\begin{array}{ccccccccccccccc}
 & & & & & & & 1 & & & & & & & \\
 & & & & & & 1 & & 1 & & & & & & \\
 & & & & & 1 & & 2 & & 1 & & & & & \\
 & & & & 1 & & 3 & & 3 & & 1 & & & & \\
 & & & 1 & & 4 & & 6 & & 4 & & 1 & & & \\
 & & 1 & & 5 & & 10 & & 10 & & 5 & & 1 & & \\
\textbf{row 6} \rightarrow & 1 & & 6 & & 15 & & \boxed{20} & & 15 & & 6 & & 1 & \ldots
\end{array}$$

$$\uparrow$$

entry 3: $\binom{6}{3} = \binom{5}{2} + \binom{5}{3} = \frac{5 \cdot 4}{1 \cdot 2} + \frac{5 \cdot 4 \cdot 3}{1 \cdot 2 \cdot 3} = 20$

(counting of rows and entries starts at 0)

Order matters, repetition allowed

How many words of length k can be formed from the letters of an n letter alphabet?

There are:

- n ways to select the first letter,
- n ways to select the second letter,
- $\dots$ and so on, n ways to select the kth letter.

So, by the product rule, the overall number is:

$$\boxed{\overbrace{n \cdot n \cdot \ldots \cdot n}^{k} = n^k}$$

Order does not matter, repetition allowed

Suppose we have an unlimited supply of 3 types of fruits:

apples (A), oranges (O), and peaches (P).

How many ways are there to select 4 pieces of fruit, if the order of selection doesn't matter, and only the type of fruit and not the individual piece matters?

Tricky. Let us try to reformulate. Suppose we have a rectangular box capable of storing 4 fruit pieces. The box has 3 compartments, for storing As, Os, and Ps. These compartments are divided by 2 movable dividers that can be shifted, depending on how many pieces of fruit of each type we want to store in the box. For example:

- 2 apples, 1 orange, 1 peach: $\boxed{A\,A \mid O \mid P}$
- 4 oranges: $\boxed{\mid O\,O\,O\,O \mid}$
- 1 apple, 3 peaches: $\boxed{A \mid\mid P\,P\,P}$
- 4 apples: $\boxed{A\,A\,A\,A \mid\mid}$

Order does not matter, repetition allowed (cont.)

So if we have to choose k objects from a set of n objects, with repetitions allowed, then we need a box with

- k places for the chosen objects, and
- $n-1$ places for the dividers dividing the box to n compartments.

$$\boxed{* * \,||\, * * * \,|\, * \,||} \qquad \text{altogether } k+n-1 \text{ places}$$

The number of ways we can choose our k objects is the number of ways we can distribute the dividers in the box: Out of the $k+n-1$ places we have to choose $n-1$ for the dividers. The number of ways doing this is:

$$\boxed{\binom{k+n-1}{n-1}} = \binom{k+n-1}{k}$$

IN THE 'FRUIT SELECTION' EXAMPLE: There are

$$\binom{4+3-1}{3-1} = \binom{6}{2} = \frac{6\,!}{4\,!\cdot 2\,!} = \frac{6\cdot 5}{1\cdot 2} = 15 \qquad \text{ways.}$$

Counting selections: a summary

The number of ways of selecting k items from a set of n items:

	Order matters: **permutations**	Order doesn't matter: **combinations**
repetitions not allowed	$n \cdot (n-1) \cdot \ldots \cdot (n-k+1)$	$\binom{n}{k}$
repetitions allowed	n^k	$\binom{k+n-1}{n-1} = \binom{k+n-1}{k}$

Exercise 4.1

How many different words can be made by reordering the letters of the word *SUCCESS* ?

SOLUTION: This word contains three *S*s, two *C*s, one *U*, and one *E*.

When we reorder these letters, the new word will also contain seven letters.

- The three *S*s can be placed among the seven positions $\binom{7}{3}$ different ways, leaving four positions free.
- Then the two *C*s can be placed in $\binom{4}{2}$ ways, leaving two free places.
- Then the *U* can be placed in $\binom{2}{1}$ ways, leaving just one position free.
- Hence *E* can be placed in $\binom{1}{1}$ way.

Therefore, by the product rule, the number of different words that can be made:

$$\boxed{\binom{7}{3} \cdot \binom{4}{2} \cdot \binom{2}{1} \cdot \binom{1}{1} = \frac{7 \cdot 6 \cdot 5}{1 \cdot 2 \cdot 3} \cdot \frac{4 \cdot 3}{1 \cdot 2} \cdot \frac{2}{1} \cdot \frac{1}{1} = 7 \cdot 6 \cdot 5 \cdot 2 = 420}$$

The Monty Hall 3-door Puzzle

You have a chance to win a large prize in a game show hosted by Monty.

The prize is behind one of 3 closed doors, and the other two doors are losers, with a goat behind each.

- You are asked to select (but not to open yet) one of the doors.
- Then Monty, who knows what is behind each door, does the following:

  Whether or not you selected the winning door, he opens one of the other two doors that he knows for sure is a losing door (selecting random if both are losing doors).
- Then he asks you whether you would like to switch doors.

Which strategy should you follow to have a better chance?

- Should you switch doors?
- Or stick to your original selection?
- Or does it not matter?

Probability theory

An **experiment** is a procedure that yields one of a given set of possible outcomes.

The **sample space** of the experiment is the set of possible outcomes:

$$S = \{s_1, s_2, \ldots, s_n\}$$

A **probability distribution** in the sample space S is a function p assigning a number $p(s_i)$ to each possible outcome s_i in S (the **probability** of s_i) such the following two conditions hold:

- $0 \le p(s_i) \le 1$, for every $i = 1, 2, \ldots, n$,
- $p(s_1) + p(s_2) + \cdots + p(s_n) \; = \; 1$.

FOR EXAMPLE: the **uniform distribution** on S

$$p(s_i) = \tfrac{1}{n}, \quad \text{for every } i = 1, 2, \ldots, n.$$

If a *fair* dice is rolled, then there are six possible equally likely outcomes: $1, 2, 3, 4, 5, 6$

So $\boxed{p(1) = p(2) = p(3) = p(4) = p(5) = p(6) = \frac{1}{6}}$

Probability distribution: two further examples

(1) If a *fair* coin is flipped, then there are two equally likely outcomes:

H (heads) and T (tails).

So $\boxed{p(H) = p(T) = \frac{1}{2}}$.

(2) But what probabilities should we assign to these outcomes when the coin is *biased* so that heads comes up twice as often as tails?

SOLUTION: We have $p(H) = 2p(T)$. Since $p(H) + p(T) = 1$ should hold, it follows that $2p(T) + p(T) = 3p(T) = 1$.

So $\boxed{p(T) = \frac{1}{3}}$ and $\boxed{p(H) = \frac{2}{3}}$.

Events and their probabilities

An **event** is a subset of the sample space, that is, a set of possible outcomes.

The **probability of an event E** is the sum of the probabilities of the outcomes in E.

FOR EXAMPLE: What is the probability that, when two fair dice are rolled, the sum of the two numbers on them is 7 ?

By the product rule, there are a total of $6 \cdot 6 = 36$ possible outcomes, each of them equally likely with a probability of $\frac{1}{36}$.

There are 6 'successful' outcomes:

$$(1,6), \quad (2,5), \quad (3,4), \quad (4,3), \quad (5,2), \quad (6,1)\,.$$

Therefore, the probability of having 7 as the sum is

$$\frac{1}{36}+\frac{1}{36}+\frac{1}{36}+\frac{1}{36}+\frac{1}{36}+\frac{1}{36} = 6\cdot\frac{1}{36} = \boxed{\frac{1}{6}}.$$

Probability of events: another example

Suppose that a dice is loaded so that 3 appears twice as often as each of the other numbers, but the other five outcomes are equally likely.

What is the probability that an odd number appears when we roll this dice?

SOLUTION: We want to find the probability $p(E)$ of the event $E = \{1, 3, 5\}$.
We know that

- $p(1) + p(2) + p(3) + p(4) + p(5) + p(6) \;=\; 1$,
- $p(1) \;=\; p(2) \;=\; p(4) \;=\; p(5) \;=\; p(6)$, and
- $p(3) \;=\; 2p(1)$.

Therefore, $p(1) \;=\; p(2) \;=\; p(4) \;=\; p(5) \;=\; p(6) \;=\; \frac{1}{7}$ and $p(3) \;=\; \frac{2}{7}$.

It follows that $\boxed{p(E) \;=\; p(1) + p(3) + p(5) \;=\; \frac{4}{7}}$.

Probability of events: yet another example

In a certain kind of lottery, people can win the top prize if they correctly choose a set of 6 numbers out of the first 59 positive natural numbers.

What is the probability of winning the top prize?

SOLUTION: The total number of ways to choose 6 numbers out of 59 is

$$\binom{59}{6} = \frac{59 \cdot 58 \cdot 57 \cdot 56 \cdot 55 \cdot 54}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6} = 45\,057\,474\,.$$

Each of these outcomes is equally likely. The event W of winning the top prize contains just one of these. Consequently,

$$\boxed{p(W) = \frac{1}{45\,057\,474} \approx 0.000000022}\,.$$

Probability of the complementary event

Let E be an event in a sample space S.
The probability of event $\bar{E}$, the **complement of E in S**,
is the sum of the probabilities of the outcomes *not* in E:

$$\boxed{p(\bar{E}) \;=\; 1 - p(E)}$$

FOR EXAMPLE: We flip a fair coin 10 times.
What is the probability that heads comes up at least once?

Let E be the event that heads comes up at least once of the 10 flips.
Then $\bar{E}$ is the event that each of the 10 times tails comes up. So

$$p(E) \;=\; 1 - p(\bar{E}) \;=\; 1 - \frac{1}{2^{10}} \;=\; 1 - \frac{1}{1024} \;=\; \frac{1023}{1024} \;\approx\; 0.999\,.$$

Probability of the union of events

$$\boxed{p(E_1 \cup E_2) \; = \; p(E_1) + p(E_2) - p(E_1 \cap E_2)}$$

FOR EXAMPLE: What is the probability that a number randomly selected from the set $X = \{n \in \mathbf{N}^+ \mid n \leq 100\}$ is divisible by either 2 or 5 or by both (by 10)?

- let E_1 be the event that the selected number in X is divisible by 2
- let E_2 be the event that the selected number in X is divisible by 5
- then $E_1 \cap E_2$ is the event that it is divisible by both 2 and 5, that is, it is divisible by 10
- as $|X| = 100$, $|E_1| = 50$, $|E_2| = 20$, and $|E_1 \cap E_2| = 10$, we obtain:

$$p(E_1 \cup E_2) = \frac{50}{100} + \frac{20}{100} - \frac{10}{100} = \frac{60}{100} = \boxed{\frac{3}{5}} = \boxed{0.6}$$

Conditional probability

We flip a fair coin 3 times. We already know how to calculate the probability of the event E that *at least 2 heads come up in a row:*

The sample space S consists of $2^3 = 8$ outcomes, each with probability $\frac{1}{8}$.

3 of them is in E: HHT, THH, HHH. So $p(E) = \boxed{\frac{3}{8}}$.

But what if the *first flip comes up heads* (event F occurs), and we are asked about the probability of E, *knowing* that F has already occurred?

Now the sample space is F: it consists of outcomes HHT, HTH, HTT, HHH

For an outcome in E to occur, it must belong to $E \cap F$: either HHT or HHH

The **conditional probability of E given F** is

$$\boxed{p(E \mid F)} = \frac{|E \cap F|}{|F|} = \frac{\frac{|E \cap F|}{|S|}}{\frac{|F|}{|S|}} = \boxed{\frac{p(E \cap F)}{p(F)}}$$

IN THE EXAMPLE:

We have $p(F) = \frac{4}{8} = \frac{1}{2}$ and $p(E \cap F) = \frac{2}{8} = \frac{1}{4}$. So $p(E \mid F) = \frac{\frac{1}{4}}{\frac{1}{2}} = \boxed{\frac{1}{2}}$.

Conditional probability: another example

We throw a pair of fair dice. What is the probability that at least one dice is a 3, given that the sum of the two dice is 5?

SOLUTION:

- let A be the event that at least one dice is a 3
- let B the event that the sum of the two dice is 5, that is, B consists of the outcomes $(1,4)$, $(2,3)$, $(3,2)$, $(4,1)$.
- as the sample space consists of $6^2 = 36$ equally probable outcomes, $p(B) = \frac{4}{36} = \frac{1}{9}$
- the event $A \cap B$ consists of two outcomes: $(2,3)$ and $(3,2)$, and so $p(A \cap B) = \frac{2}{36} = \frac{1}{18}$

Therefore,

$$\boxed{p(A \mid B) = \frac{p(A \cap B)}{p(B)} = \frac{\frac{1}{18}}{\frac{1}{9}} = \frac{1}{2}}$$

Observe that the unconditional probability of A is different: $p(A) = \frac{11}{36}$

Conditional probability: yet another example

We flip a fair coin 3 times. What is the probability that an odd number of tails appear, knowing that the first flip comes up tails?

SOLUTION:

- let X be the event that <u>an odd number of tails appear</u>
- let Y be the event that <u>the first flip comes up tails</u>
- as the sample space consists of $2^3 = 8$ equally probable outcomes, $p(Y) = \frac{4}{8} = \frac{1}{2}$
- the event $X \cap Y$ consists of two outcomes: THH and TTT, and so $p(X \cap Y) = \frac{2}{8} = \frac{1}{4}$

Therefore,

$$\boxed{p(X \mid Y) = \frac{p(X \cap Y)}{p(Y)} = \frac{\frac{1}{4}}{\frac{1}{2}} = \frac{1}{2}}$$

Observe that this time the unconditional probability of X is the same: $p(X) = \frac{4}{8} = \frac{1}{2}$

Independence

When two events E and F are **independent,** the occurrence of one of the events gives no information about the probability of the other event:

$$\boxed{p(E \mid F) \;=\; p(E)} \qquad \text{or equivalently,} \qquad \boxed{p(F \mid E) \;=\; p(F)}$$

Using that $p(E \mid F) \;=\; \frac{p(E\cap F)}{p(F)}$, we obtain that E and F are independent events, whenever

$$\boxed{p(E \cap F) \;=\; p(E) \cdot p(F)}$$

FOR EXAMPLE:

- Events E and F on lecture slide 143 are not independent.
- Events A and B on lecture slide 144 are not independent.
- Events X and Y on lecture slide 145 are independent.

Bayes' Theorem

This might be a very useful tool when

- E and F are events from the same sample space S
- we can easily compute $p(E \mid F)$
- but we want to know $\underline{p(F \mid E)}$

$$\boxed{p(F \mid E) \;=\; \frac{p(E \mid F)p(F)}{p(E \mid F)p(F) \;+\; p(E \mid \bar{F})p(\bar{F})}}$$

Applying Bayes' Theorem: An example

A frog's climbing out of a well is affected by the weather. When it rains, he falls back down the well with a probability of $\frac{1}{10}$. In dry weather, he only falls back down with probability of $\frac{1}{25}$. The probability of rain is $\frac{1}{5}$.

If we know that the frog fell back, what is the probability that it was a rainy day?

SOLUTION: Let $\boxed{F}$ be the event that *the frog fell back,* and

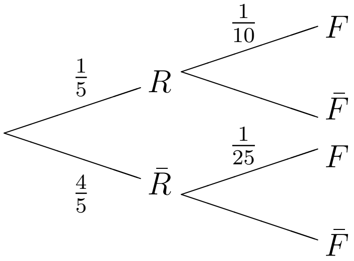
$\boxed{R}$ be the event that *it was a rainy day.*

We want to find $p(R \mid F)$.

- $\boxed{p(R) = \frac{1}{5}}$ $\boxed{p(\bar{R}) = 1 - \frac{1}{5} = \frac{4}{5}}$
- $\boxed{p(F \mid R) = \frac{1}{10}}$ $\boxed{p(F \mid \bar{R}) = \frac{1}{25}}$

$$p(R \mid F) = \frac{p(F \mid R)p(R)}{p(F \mid R)p(R) + p(F \mid \bar{R})p(\bar{R})} =$$

$$\frac{\frac{1}{10} \cdot \frac{1}{5}}{\frac{1}{10} \cdot \frac{1}{5} + \frac{1}{25} \cdot \frac{4}{5}} = \frac{\frac{1}{50}}{\frac{1}{50} + \frac{4}{125}} = \frac{\frac{25}{1250}}{\frac{25}{1250} + \frac{40}{1250}} = \frac{5}{13}$$



Applying Bayes' Theorem: Example 2

There are two boxes, Box_1 and Box_2.

- Box_1 contains 2 green balls and 7 red balls
- Box_2 contains 4 green balls and 3 red balls

Bob is blindfolded, and selects a ball

- by randomly choosing one of the boxes first,
- then randomly choosing a ball from it.

If we know that Bob has selected a red ball, what is the probability that he selected a ball from Box_1?

Applying Bayes' Theorem: Example 2 (cont.)

If we know that Bob has selected a red ball, what is the probability that he selected a ball from *Box*$_1$?

SOLUTION: Let $\boxed{R}$ be the event that *Bob has chosen a red ball*, and

$\boxed{F}$ be the event that *Bob has chosen a ball from Box*$_1$.

We want to find $p(F \mid R)$.

- $p(R \mid F) = \frac{7}{9}$. as *Box*$_1$ contains 2 green balls and 7 red balls
- $\boxed{\bar{F}}$: is the event that *Bob has chosen a ball from Box*$_2$.

  Then $p(F) = p(\bar{F}) = \frac{1}{2}$.
- $p(R \mid \bar{F}) = \frac{3}{7}$. as *Box*$_2$ contains 4 green balls and 3 red balls

Then by Bayes' Theorem:

$$p(F \mid R) = \frac{\frac{7}{9} \cdot \frac{1}{2}}{\frac{7}{9} \cdot \frac{1}{2} + \frac{3}{7} \cdot \frac{1}{2}} = \frac{\frac{7}{9}}{\frac{49+27}{63}} = \frac{\frac{7}{9}}{\frac{76}{63}} = \frac{49}{76} \approx 0.645$$

Applying Bayes' Theorem: Example 3

Suppose that 1 person in 100 000 has a particular rare disease for which there is a quite accurate diagnostic test:

- It is correct 99% of the time when given to someone with the disease.
- It is correct 99.5% of the time when given to someone who does not have the disease.

What is the probability that <u>someone who tests positive for the disease actually has the disease?</u>

SOLUTION: Let $\boxed{D}$ be the event that a person has the disease, and $\boxed{T}$ be the event that a person tests positive for the disease.

- $p(T \mid D) \; = \; 0.99$
- $p(\bar{T} \mid \bar{D}) \; = \; 0.995$, so $p(T \mid \bar{D}) \overset{Ex.5.5}{=} 1 - p(\bar{T} \mid \bar{D}) \; = \; 1 - 0.995 \; = \; 0.005$
- $p(D) \; = \; \frac{1}{100\,000} \; = \; 0.00001$, so $p(\bar{D}) \; = \; 1 - 0.00001 \; = \; 0.99999$
- $\underline{p(D \mid T)} \; = \; \frac{0.99 \cdot 0.00001}{0.99 \cdot 0.00001 + 0.005 \cdot 0.99999} \; \approx \; \underline{0.002}$

Bayesian spam filters

A Bayesian spam filter uses information about previously seen e-mail messages to guess whether an incoming e-mail is spam. It looks for occurrences of particular words in messages. For a word w, the probability that w appears in a spam message is estimated by determining

- the number of times w appears in a message from a large set of messages *known to be spam*, and
- the number of times w appears in a large set of messages that are *known not to be spam*.

Unfortunately, spam filters sometimes fail to identify a spam message as spam: this is called a **false negative**. And they sometimes identify a message that is not spam as spam: this is called a **false positive**.

When testing for spam, it is important to minimise false positives, because filtering out wanted e-mail is much worse than letting some spam through.

Bayesian spam filters: an example

Suppose we have found that the word "*Rolex*" occurs in 250 of 2000 messages known to be spam, and in 5 of 1000 messages known not to be spam.

Estimate the probability that an incoming message containing the word "*Rolex*" is spam, assuming that it is equally likely that an incoming message is spam or not spam. If our threshold for rejecting a message as spam is 0.9 , will we reject this message?

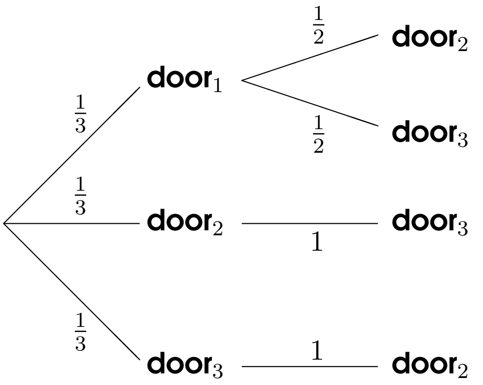
SOLUTION: Let S be the event that an incoming message is spam, and R be the event that the message contains the word "*Rolex*". Then:

- $p(R \mid S) = \frac{250}{2000} = 0.125$
- $p(R \mid \bar{S}) = \frac{5}{1000} = 0.005$
- $p(S) = p(\bar{S}) = 0.5$
- $\boxed{p(S \mid R) = \frac{0.125 \cdot 0.5}{0.125 \cdot 0.5 + 0.005 \cdot 0.5} = \frac{0.125}{0.13} \approx 0.962}$

As $0.962 > 0.9$, we would reject the message as spam.

The 3-door puzzle revisited: conditional probability directly

You selected a door: let's call it **door**$_1$. Then Monty opened another door:



Prize location:		Monty opens:	Overall probability:	What you get if you stick:	switch:
$\frac{1}{3}$ **door**$_1$	$\frac{1}{2}$	**door**$_2$	$\frac{1}{3}\cdot\frac{1}{2}=\frac{1}{6}$	prize	goat
	$\frac{1}{2}$	**door**$_3$	$\frac{1}{3}\cdot\frac{1}{2}=\frac{1}{6}$	prize	goat
$\frac{1}{3}$ **door**$_2$	1	**door**$_3$	$\frac{1}{3}\cdot 1=\frac{1}{3}$	goat	prize
$\frac{1}{3}$ **door**$_3$	1	**door**$_2$	$\frac{1}{3}\cdot 1=\frac{1}{3}$	goat	prize

As $\frac{1}{3}+\frac{1}{3}=\frac{2}{3}>\frac{1}{3}=\frac{1}{6}+\frac{1}{6}$, you'll have a better chance if you switch from **door**$_1$ to the door Monty did not open.

The 3-door puzzle revisited: using Bayes theorem

- You selected a door: let's call it **door**$_1$
- Then Monty opened another door: let's call it **door**$_2$
- Let's call the remaining 3rd door **door**$_3$

Consider the following events:

W_1: **door**$_1$ is the winning door	W_2: **door**$_2$ is the winning door
W_3: **door**$_3$ is the winning door	M_2: Monty opened **door**$_2$

We are interested in $\boxed{p(\overline{W_1} \mid M_2)}$:

- if $p(\overline{W_1} \mid M_2) > \frac{1}{2}$ then you should switch from **door**$_1$ to **door**$_3$
- if $p(\overline{W_1} \mid M_2) < \frac{1}{2}$ then you should stick to **door**$_1$
- and if $p(\overline{W_1} \mid M_2) = \frac{1}{2}$ then it doesn't matter

The 3-door puzzle revisited: using Bayes theorem (cont.)

We want to use Bayes' theorem to compute $p(\overline{W_1} \mid M_2)$:

$$p(\overline{W_1} \mid M_2) \;=\; \frac{p(M_2 \mid \overline{W_1}) \cdot p(\overline{W_1})}{p(M_2 \mid \overline{W_1}) \cdot p(\overline{W_1}) + p(M_2 \mid W_1) \cdot p(W_1)}$$

So we need to compute:

- $p(W_1)$
- $p(\overline{W_1})$
- $p(M_2 \mid W_1)$
- $p(M_2 \mid \overline{W_1})$

The 3-door puzzle revisited: using Bayes theorem (cont.)

- $\underline{p(W_1)} = p(W_2) = p(W_3) = \frac{1}{3}$

  These are the 3 different outcomes and they are equally likely.

- $\underline{p(\overline{W_1})} = 1 - p(W_1) = 1 - \frac{1}{3} = \frac{2}{3}$

- $\underline{p(M_2 \mid W_1)} = \frac{1}{2}$

  If your selected **door**$_1$ is the winning one, then Monty opens randomly one of the other two doors.

- $\underline{p(M_2 \mid \overline{W_1})} = \dfrac{p(M_2 \cap \overline{W_1})}{p(\overline{W_1})}$

  $p(M_2 \cap \overline{W_1}) = p\big(M_2 \cap (W_2 \cup W_3)\big) = p\big((M_2 \cap W_2) \cup (M_2 \cap W_3)\big) = p(M_2 \cap W_3)$

  $p(M_2 \cap W_3) = p(M_2 \mid W_3) \cdot p(W_3) = 1 \cdot \frac{1}{3} = \frac{1}{3}$

  $p(M_2 \mid W_3) = 1$ because if **door**$_3$ is the winning door and you selected **door**$_1$, then Monty surely opens **door**$_2$ (as he never opens the winning door).

- $p(M_2 \mid \overline{W_1}) = \dfrac{p(M_2 \cap \overline{W_1})}{p(\overline{W_1})} = \dfrac{\frac{1}{3}}{\frac{2}{3}} = \frac{1}{2}$

The 3-door puzzle revisited: the conclusion by Bayes' theorem

So we have: $p(W_1) = \frac{1}{3}$ $\quad p(\overline{W_1}) = \frac{2}{3}$ $\quad p(M_2 \mid W_1) = \frac{1}{2}$ $\quad p(M_2 \mid \overline{W_1}) = \frac{1}{2}$

Now we can apply Bayes' theorem:

$$\boxed{p(\overline{W_1} \mid M_2)} = \frac{p(M_2 \mid \overline{W_1}) \cdot p(\overline{W_1})}{p(M_2 \mid \overline{W_1}) \cdot p(\overline{W_1}) + p(M_2 \mid W_1) \cdot p(W_1)}$$

$$= \frac{\frac{1}{2} \cdot \frac{2}{3}}{\frac{1}{2} \cdot \frac{2}{3} + \frac{1}{2} \cdot \frac{1}{3}} = \frac{\frac{2}{6}}{\frac{3}{6}} = \boxed{\frac{2}{3}}$$

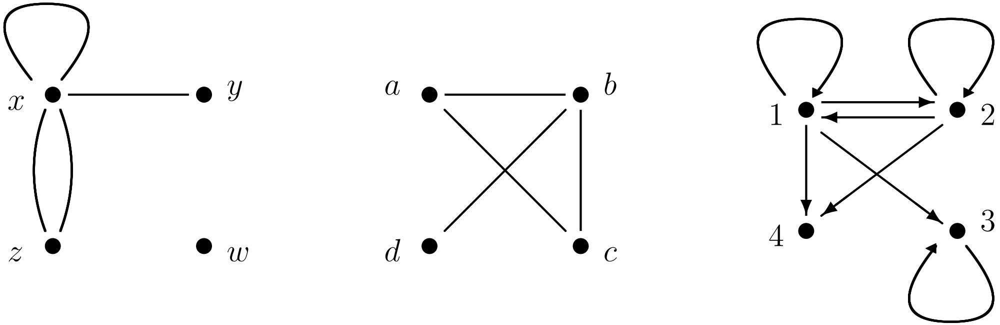
Therefore:

$$\boxed{p(\overline{W}_1 \mid M_2) = \frac{2}{3} > \frac{1}{2}}$$

So you'll have a <u>better chance if you switch</u> from **door**$_1$ to **door**$_3$.

Graphs

Graphs are drawings with dots and (not necessarily straight) lines or arrows.



The dots are called **vertices** (or **nodes**).

The lines or arrows are called **edges**.

Different kinds of graphs

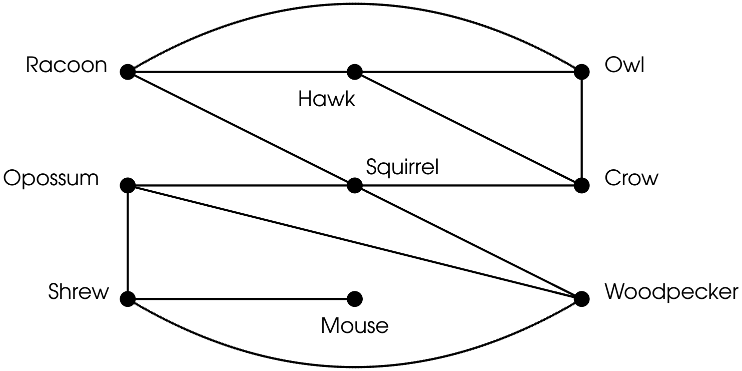
Type	Edges	Multiple edges	Loop edges
(simple) graph	undirected	no	no
multigraph	undirected	yes	yes
directed graph	directed	no	yes
. . .	. . .	. . .	. . .

Because graphs have applications in a variety of disciplines, many different terminologies of graph theory have been introduced. You may find different ones in different books, areas, etc.

Example 1: Niche overlap graphs in ecology

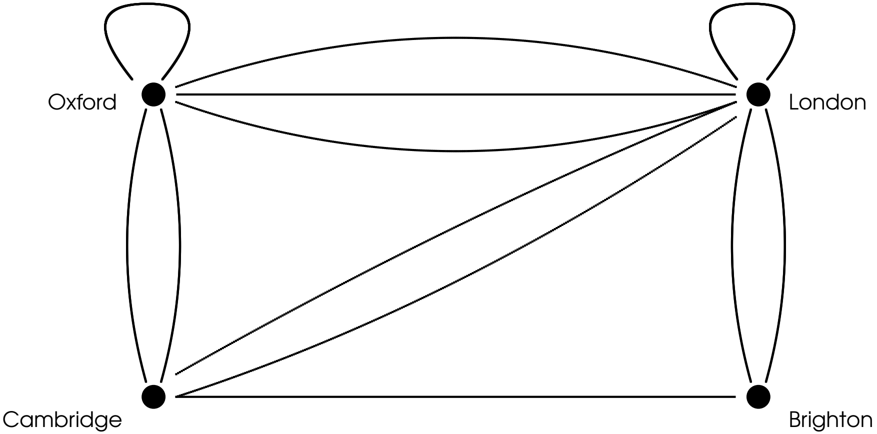
Competitions between species in an ecosystem can be modelled using a **niche overlap graph**:

Each species is represented by a vertex. An edge connects two vertices if the two species represented by these vertices compete (that is, some of the food resources they use are the same).



⇝ simple graph

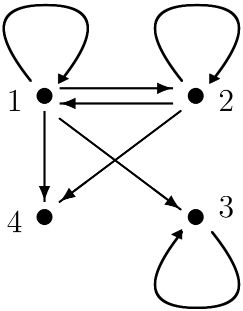
Example 2: Road networks



$\rightsquigarrow$ multigraph

Example 3: Representing binary relations

$$R = \{(1,1),(1,2),(1,3),(1,4),(2,1),(2,2),(2,4),(3,3)\}.$$



$\rightsquigarrow$ directed graph

Undirected graphs: basic terminology

If there is an edge e between vertices u and v, we say that

- u and v are **adjacent**, and
- e is **incident with** u and v.

The **degree** of a vertex is the number of edges incident with it.

- A vertex of degree zero is called **isolated**.

  So an isolated vertex is not adjacent to any vertex.

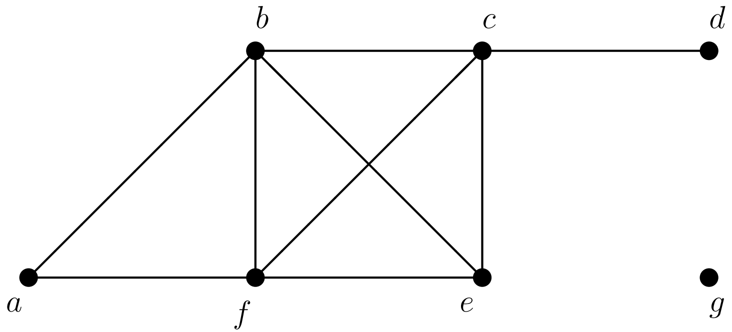
- A vertex of degree one is called **pendant**.

  So a pendant vertex is adjacent to exactly one other vertex.

$\boxed{\text{Handshaking theorem:}}$

$$\text{number of edges} = \frac{\text{sum of the degrees of vertices}}{2}$$

Degrees of vertices: example 1



- degree$(a) = 2$,
- degree$(b) =$ degree$(c) =$ degree$(f) = 4$,
- degree$(e) = 3$,
- degree$(d) = 1$, so d is pendant,
- degree$(g) = 0$, so g is isolated.

Directed graphs: basic terminology

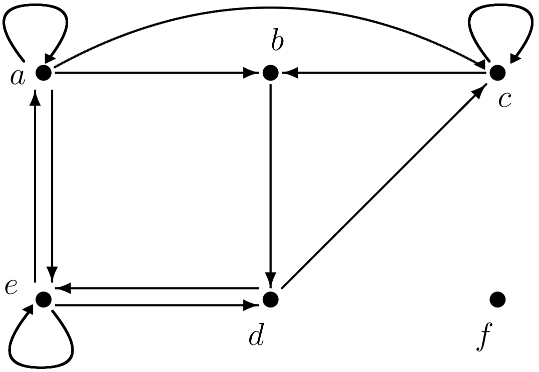
If there is an edge e going from vertex u to v, we say that

- u is **adjacent to** v,
- u is the **initial** or **start vertex** of e, and
- v is the **terminal** or **end vertex** of e.

The **in-degree** of a vertex v is the number of edges with v as their terminal vertex. The **out-degree** of a vertex v is the number of edges with v as their initial vertex. (A loop at a vertex contributes 1 to both the in-degree and the out-degree of this vertex.)

$$\begin{aligned}\text{number of edges} &= \text{sum of the in-degrees of vertices}\\ &= \text{sum of the out-degrees of vertices.}\end{aligned}$$

Degrees of vertices: example 2



- in-degree(a) = in-degree(b) = in-degree$(d) = 2$,

  in-degree(c) = in-degree$(e) = 3$,

  in-degree$(f) = 0$,

- out-degree$(a) = 4$, out-degree$(b) = 1$,

  out-degree(c) = out-degree$(d) = 2$, out-degree$(e) = 3$,

  out-degree$(f) = 0$.

Representing graphs: adjacency matrix

- List the vertices in some order horizontally from left to right.
- Then, using the same order, list them vertically from top to bottom.
- The entry in the i^{th} row and the j^{th} column is the

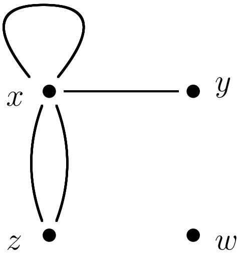
  number of edges going from vertex i to vertex j.

If the graph is undirected, then

the number in the i^{th} row and j^{th} column

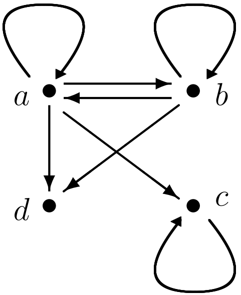
$=$ the number in the j^{th} row and i^{th} column.

Adjacency matrices: examples



	x	y	w	z
x	1	1	0	2
y	1	0	0	0
w	0	0	0	0
z	2	0	0	0

	y	x	w	z
y	0	1	0	0
x	1	1	0	2
w	0	0	0	0
z	0	2	0	0



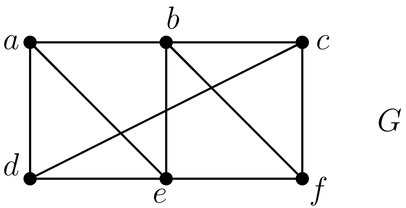
	a	b	c	d
a	1	1	1	1
b	1	1	0	1
c	0	0	1	0
d	0	0	0	0

	b	c	d	a
b	1	0	1	1
c	0	1	0	0
d	0	0	0	0
a	1	1	1	1

Paths in simple graphs

- A **path** is a sequence of vertices travelling along edges.
- The **length** of a path is the *number of edges* in it.
- A path is called **simple** if it does not contain the same *edge* twice.
- A **Hamiltonian path** is a *simple* path passing through *every vertex exactly once.*

FOR EXAMPLE:



(a, b, c, f, b, e) is a simple path in G of length 5

(d, c, a, e) is not a path in G — WHY? no edge between c and a

(a, b, e, d, a, b) is a path in G of length 5, but it is not simple
— WHY? contains the edge between a and b twice

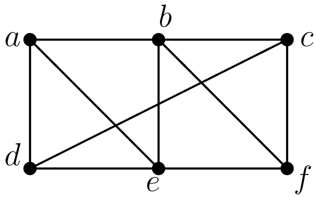
(d, a, e, b, f, c) is a Hamiltonian path in G

(a, b, c, f, b, e, d) is a simple path, but not a Hamiltonian path in G
— WHY? passes b more than once

Cycles in simple graphs

- A **cycle** is a path beginning and ending with the same vertex.
- The **length** of a cycle is the *number of edges* in it.
- A cycle is called **simple** if it does not contain the same *edge* twice.
- A **Hamiltonian cycle** is a *simple* cycle passing through *every vertex* *exactly once.*

FOR EXAMPLE:



G

(a, d, c, b, a) is a simple cycle in G of length 4

(b, e, d, a, e, b) is a cycle in G of length 5, but it is not simple

— WHY? contains the edge between b and e twice

(c, f, e, d, a, b, c) is a Hamiltonian cycle in G

(e, d, c, f, e, b, a, e) is a simple cycle in G, but not Hamiltonian

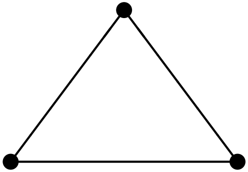
— WHY? passes e more than once

(a, d, e, f, b, a) is a simple cycle in G, but not Hamiltonian

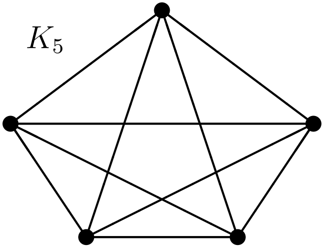
— WHY? does not pass c

Special simple graphs

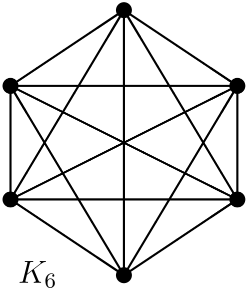
- The **complete graph on n vertices** (or n-**clique**), denoted by $\boxed{K_n}$, is the simple graph that contains an edge between each pair of distinct vertices.



K_3

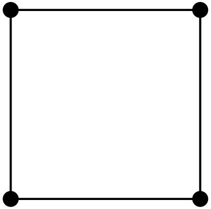


K_5

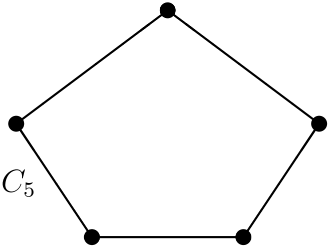


K_6

- The n-**cycle** is denoted by $\boxed{C_n}$, for $n \geq 4$.



C_4



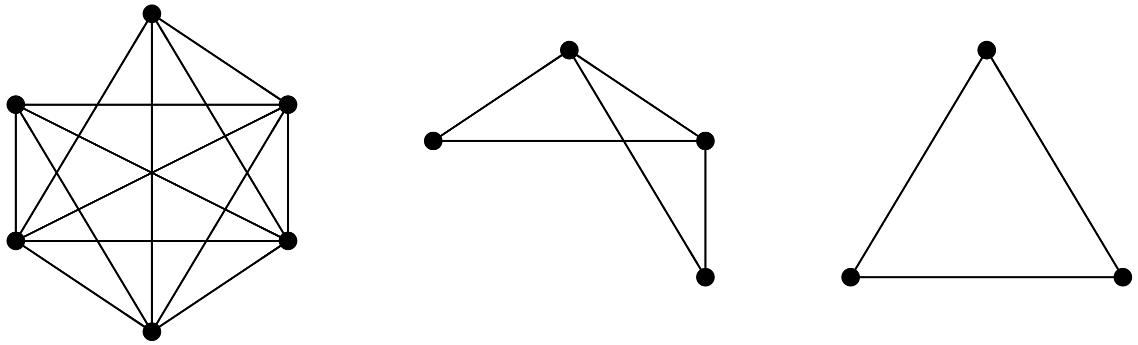
C_5

Subgraphs of graphs

When edges and vertices are removed from a graph, without removing endpoints of any remaining edges, a smaller graph is obtained.

Such a graph is called a **subgraph** of the original graph.

FOR EXAMPLE: Each of the following 3 graphs

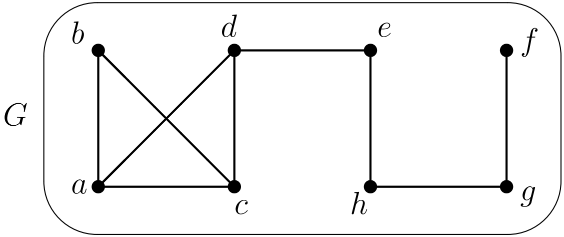


is a subgraph of K_6.

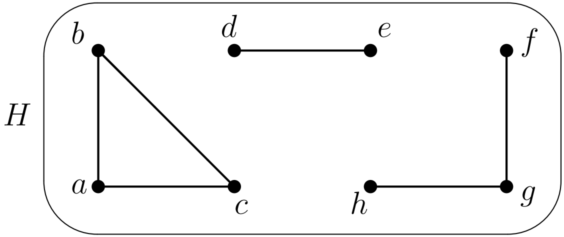
Connected graphs

A simple graph is called **connected** if there is a path between every pair of distinct vertices.

FOR EXAMPLE:



is connected

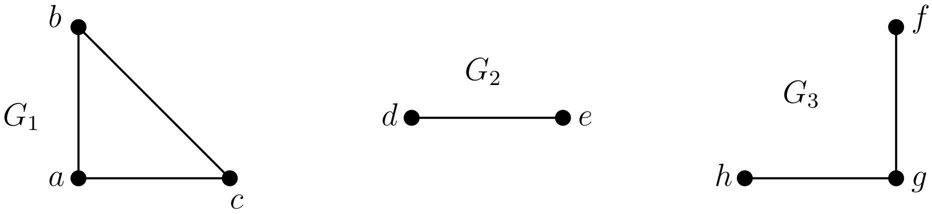


is not connected

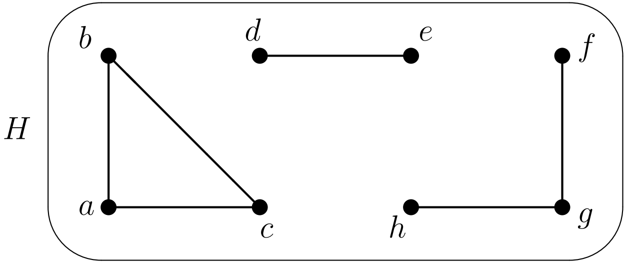
Connected components of graphs

A **connected component** of a graph is a maximal connected subgraph.

- If a graph is connected, then it has only 1 connected component, *itself.*
- But if it is not connected, it can have more:



are the 3 connected components of

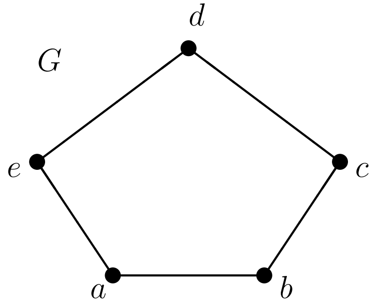


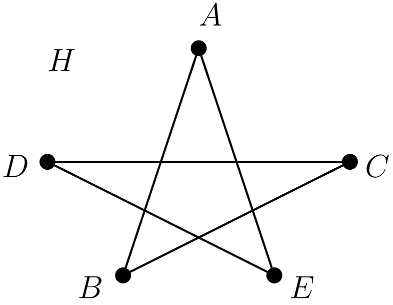
Isomorphism of graphs: an example

The following instructions were told to two persons:

> "Draw and label five vertices with a, b, c, d, and e.
> Connect a and b, b and c, c and d, d and e, and a and e."

They drew the graphs:



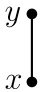


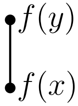
Surely, these drawings describe the same situation, though the graphs G and H appear dissimilar.

Isomorphism of graphs

Graphs G and H are <u>**isomorphic**</u> if there is an <u>**isomorphism**</u> between them:
A function f from the vertices of G to the vertices of H such that

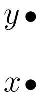
- f is a bijection (one-to-one and onto)
- f 'takes' edges to edges:

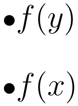
  for all vertices x, y in G, if 

then 

in H

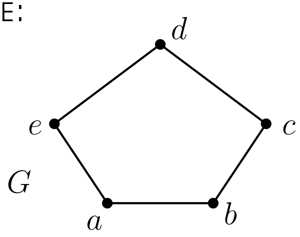
- f 'takes' non-edges to non-edges:

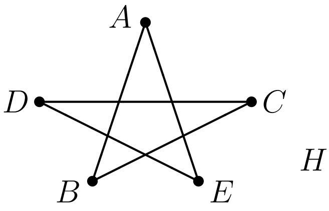
  for all vertices x, y in G, if 

then 

in H

FOR EXAMPLE:



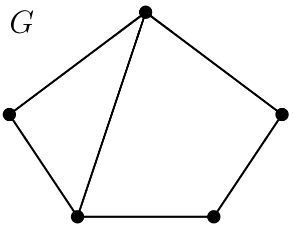


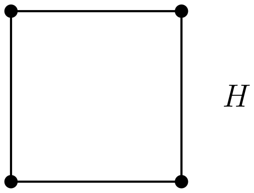
The function f defined by taking

$$f(a) = A, \qquad f(b) = B, \qquad f(c) = C, \qquad f(d) = D, \qquad f(e) = E$$

is an isomorphism, showing that graphs G and H are isomorphic.

Isomorphic or not — how can we decide?





TASK: Determine whether two graphs G and H are isomorphic or not.

isomorphic = there is an isomorphism
not isomorphic = there is no isomorphism

- We can try **all possible functions** from G to H, and check whether any of them is an isomorphism.
- But this might take a lot of time: there are $4^5 = 1024$ possible functions even for this simple example (see lecture slide 119).
  So for larger graphs it is kind of hopeless.

Is there some quicker way?

Isomorphism of graphs: invariants

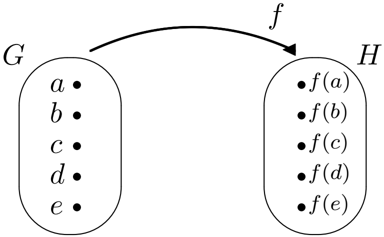
A property $\mathcal{P}$ of graphs is called an **invariant** if it is 'preserved under isomorphisms': If G and H are isomorphic graphs, and G has property $\mathcal{P}$,

then H has property $\mathcal{P}$ as well.

FOR EXAMPLE: "*Having* 5 *vertices*" is an invariant:
If G and H are isomorphic graphs, and G has 5 vertices,
then H has 5 vertices as well.

WHY? If G and H are isomorphic, then there is an isomorphism f between them.

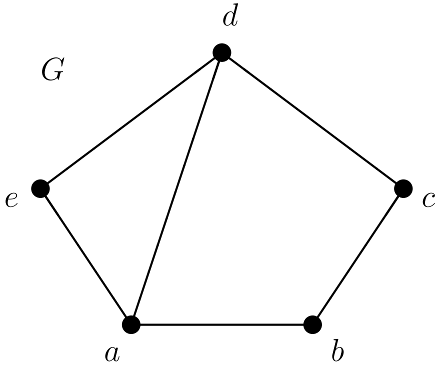
- f is a *bijection* from the vertices of G (domain) to the vertices of H (codomain).

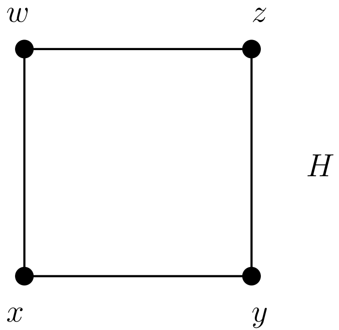


- As f is *one-to-one,* H has at least 5 vertices.
- And as f is *onto,* H has at most 5 vertices.

Exercise 6.1

Determine whether G and H are isomorphic or not:





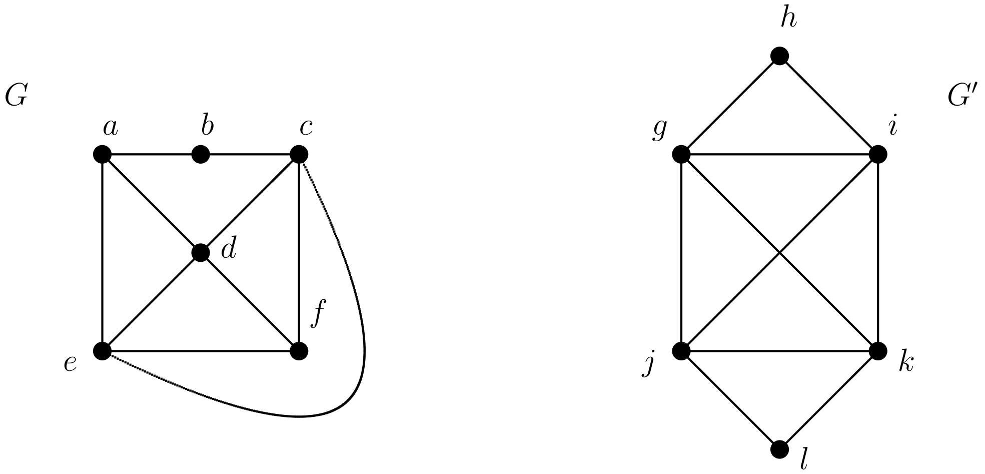
SOLUTION: We’ve just seen that the property “having 5 vertices” is an invariant. This property holds for G, but not for H. Therefore, G and H are not isomorphic.

Some more examples of invariants

- the number of edges
- the number of vertices of each degree
- containing a triangle (K_3) as a subgraph
- containing two K_4s as disjoint subgraphs
- containing a simple cycle of length 4
- having a path of length 2 between two vertices of degree 2
- $\dots$

Exercise 6.2

Determine whether G and G' are isomorphic or not:

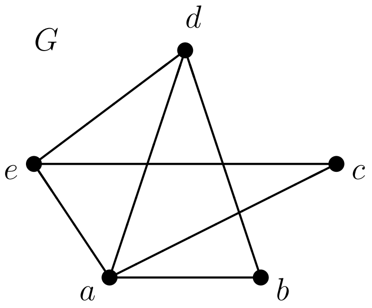


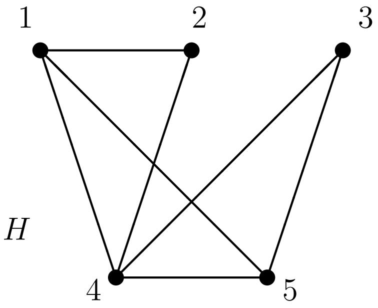
SOLUTION: Now it is a bit harder to find an invariant. Both graphs have 6 vertices and 10 edges. But, say, "having a vertex of degree 3" is an **invariant.**

This property holds in G (say, degree$(a) = 3$), but does not hold in G', showing that G and G' are not isomorphic.

Exercise 6.3

Determine whether G and H are isomorphic or not:





SOLUTION: The function f defined by taking

$$f(a) = 4, \qquad f(b) = 2, \qquad f(c) = 3, \qquad f(d) = 1, \qquad f(e) = 5$$

is an isomorphism (because f is a bijection, takes edges to edges, and non-edges to non-edges). This shows that graphs G and H are isomorphic.

HOW TO FIND AN ISOMORPHISM?

Hint: Always keep in mind that if h is an isomorphism between graphs G and H, then for every vertex x in G, the degrees of x and $h(x)$ must be the same.

Isomorphic or not — how to decide?

TASK: Determine whether two graphs G and H are isomorphic or not.

SOLUTION: There is no quick and easy way, without thinking.

The computational complexity of the *graph isomorphism problem* is one of the most famous unsolved problems in computer science.

It is **not known**

- whether the problem is *solvable in polynomial time*
  (that is, whether there exists a general method that is always quicker than trying out all possible functions), and
- whether the problem is *definitely NOT solvable in polynomial time*.
  (For many other well-known computational problems, there are known PROOFS showing that they cannot be solved quicker than trying out all options.)

Isomorphic or not — so how can we decide?

TASK: Determine whether two graphs G and H are isomorphic or not.

SOLUTION: We have to try in parallel:

- To describe a bijection between the vertices of G and H that 'takes' edges to edges, non-edges to non-edges.

  $\rightsquigarrow$ If we succeed, the answer is $\boxed{\text{YES}}$

- To find an invariant $\mathcal{P}$ and show that G has $\mathcal{P}$ but H doesn't, or the other way round.

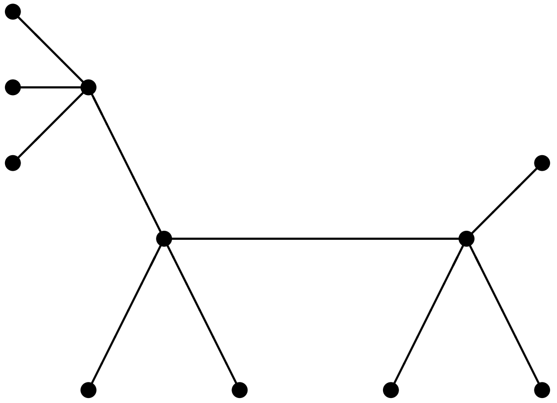
  $\rightsquigarrow$ If we succeed, the answer is $\boxed{\text{NO}}$

It would be nice to have a list of easily checkable invariants that isomorphic graphs and *only* isomorphic graphs share. Then we would just have to check those. BTW having such a list would solve the *graph isomorphism problem*.

Unfortunately, no one has yet succeeded in finding such a list of invariants, so determining whether two graphs are isomorphic or not might require some **creative thinking.**

Special graphs: trees

A **tree** is a connected simple graph with no simple cycles.

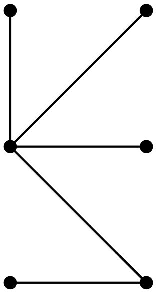


Some useful facts about trees:

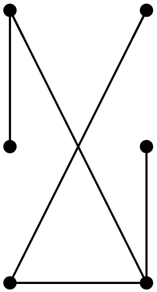
- In a tree there is a unique simple path between any two of its vertices.
- If we add an edge to a tree, it creates a cycle.
- If we remove an edge from a tree, it becomes not connected.

Trees: examples and non-examples

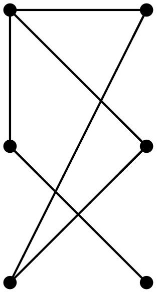
Trees:



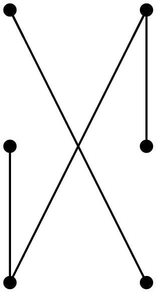
and



Not trees:

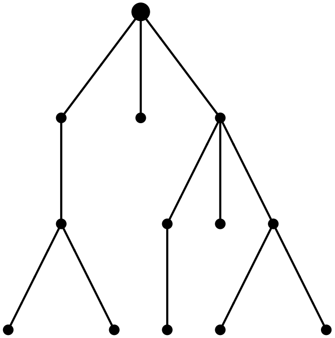


and



Rooted trees

A **rooted tree** is a tree in which one vertex has been designated as the root. We can change an unrooted tree to a rooted tree by choosing *any* vertex as the root. We usually draw a rooted tree with its root at the top:



Two rooted trees are **isomorphic** if there is a bijection between their vertices that

- takes the root to root, and
- takes edges to edges, and non-edges to non-edges.

Rooted trees: basic terminology

The terminology for trees has botanical and genealogical origins.

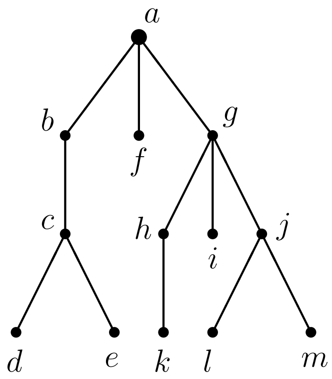
- If vertices u and v are connected by an edge, and u is closer to the root than v (that is, above v), then

  - u is called the **parent** of v, and v is called a **child** of u.

  Vertices with the same parent are called **siblings**.

- A childless vertex is called a **leaf**.
- Vertices with at least one child are called **internal**.
- The **ancestors** of a non-root vertex v are the vertices in the (unique) simple path from the root to v.
- The **descendants** of vertex v are those vertices that have v as an ancestor.

Basic terminology: an example



- The root is a.
- The parent of c is b.
- The children of g are h, i, and j.
- The siblings of h are i and j.
- The ancestors of e are c, b, and a.
- The descendants of b are c, d, and e.
- The internal vertices are a, b, c, g, h, and j.
- The leaves are d, e, f, i, k, l, and m.

Applications of trees

Trees are used for modelling and problem solving in a wide variety of disciplines.

FOR EXAMPLE:

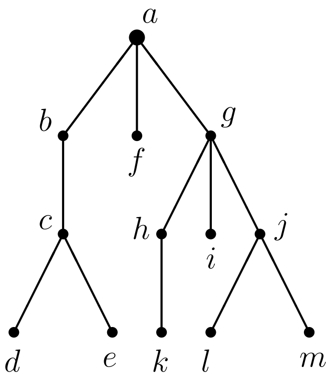
- family trees in genealogy
- representing organisations
- computer file systems
- constructing efficient methods for locating items in a list: *binary search trees*
- game trees to analyse winning strategies in games
- decision trees
- *decomposition trees* to parse arithmetical and logic formulas and expressions
- . . .

Rooted trees: the level of a vertex

The **level** (or **depth**) of a vertex v is the length of the (unique) path from the root to v.

The level of the root is 0.

FOR EXAMPLE:



level of a is 0

level of f is 1

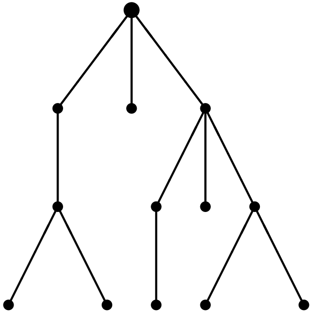
level of j is 2

level of e is 3

Rooted trees: height

The **height** of a rooted tree is the maximum of the levels of its vertices.

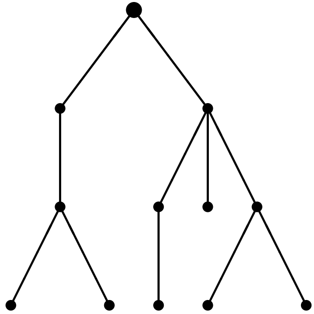
FOR EXAMPLE:

height of  is 3

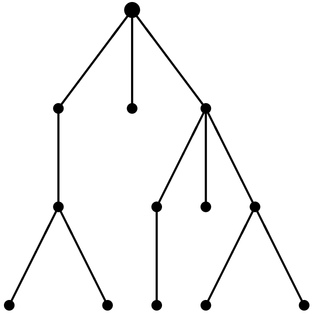
Balanced rooted trees

A rooted tree of height n is called **balanced** if all its leaves are of level n or $n-1$.

FOR EXAMPLE:



balanced

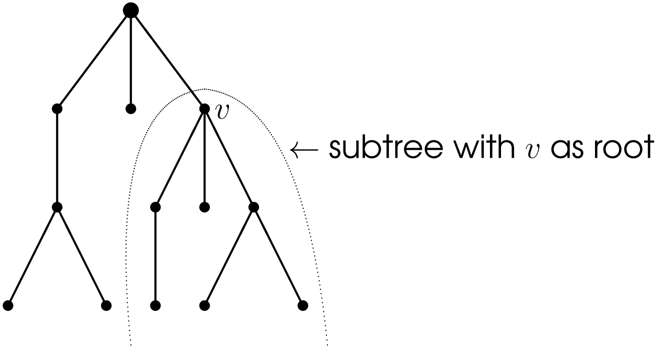


not balanced

Rooted trees: subtrees

- If v is a vertex in a rooted tree T, the **subtree** with v as its root is the subgraph of T consisting of v, all its descendants, and all edges incident to these descendants.

FOR EXAMPLE:



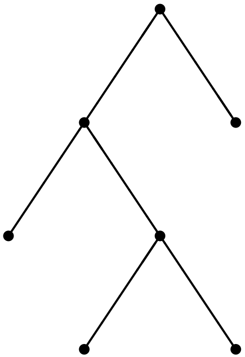
Special trees

- A rooted tree is called an **m-ary tree** if every internal vertex has no more than m children.

- A rooted tree is called a **full m-ary tree** if every internal vertex has exactly m children.

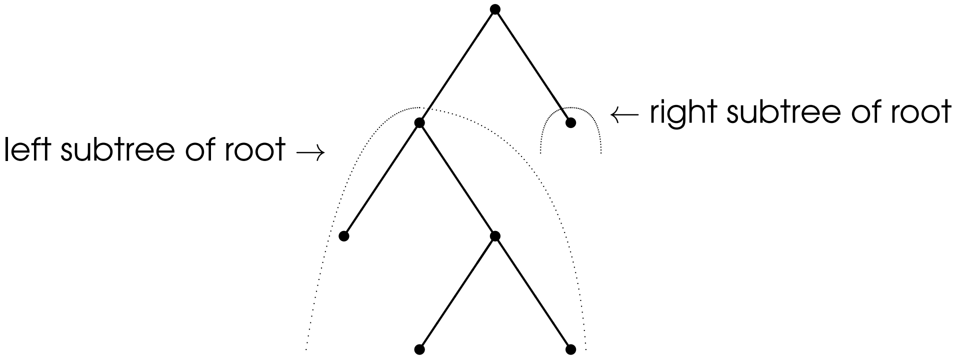
  A rooted tree is called a **full binary tree** if every internal vertex has exactly 2 children: a **left child** and a **right child.**

FOR EXAMPLE: A full binary tree:



Useful observations about full binary trees

Recall: A full binary tree is a rooted tree in which every internal vertex has exactly 2 children.



When we have a full binary tree of height n then

- the left and right subtrees of the root are **both** full binary trees of $\underline{\text{height } \leq n-1}$
- **at least one** of the left and right subtrees of the root is a full binary tree of $\underline{\text{height } n-1}$ (but not necessarily both)

Counting vertices and edges of trees

- A full binary tree with n internal vertices contains $2n+1$ vertices altogether.

  WHY? Every vertex, except the root, is the child of an internal vertex.

  Because each of the n internal vertices has 2 children, there are $2n$ vertices in the tree other than the root.

- A full m-ary tree with n internal vertices contains $m \cdot n+1$ vertices altogether.

  WHY? Every vertex, except the root, is the child of an internal vertex.

  Because each of the n internal vertices has m children, there are $m \cdot n$ vertices in the tree other than the root.

Exercise 7.1

Prove by induction that, for every positive integer n, every full binary tree of height $\leq n$ has $\leq 2^n$ leaves.

$P(n)$: the number of leaves of any full binary tree of height $\leq n$ is $\leq 2^n$

SOLUTION: **Basis step:** We need to prove

$P(1)$: the number of leaves of any full binary tree of height ≤ 1 is $\leq 2^1$

So let's see. A full binary tree of height ≤ 1 is either of height 0, or of height 1.

- A full binary tree of height 0 is just a root, so it has 1 leaf and $1 \leq 2 = 2^1$.
- And a full binary tree of height 1 consists of a root and its two children, so it has 2 leaves and $2 \leq 2^1$.

Exercise 7.1 (cont.)

Inductive step: We need show that, for all positive integer k,

if $P(k)$ holds then $P(k+1)$ holds as well.

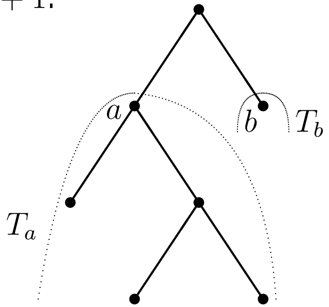
So suppose that for some positive integer k,

the number of leaves in any full binary tree of height $\leq k$ is $\leq 2^k$ **(IH).**

We need to show that

the number of leaves in any full binary tree of height $\leq k+1$ is $\leq 2^{k+1}$.

So let T be an arbitrary full binary tree of height $\leq k+1$.



- Take the two children of the root of T, say, a and b. Let T_a denote the subtree with root a, and T_b denote the subtree with root b. Then both T_a and T_b are full binary trees of height $\leq k$ (see lecture slide 197).
- Thus, by the IH, T_a has $\leq 2^k$ leaves, and T_b has $\leq 2^k$ leaves as well.
- As the leaves of T consists of all the leaves in T_a plus all the leaves in T_b, T has $\leq 2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$ leaves, as required.

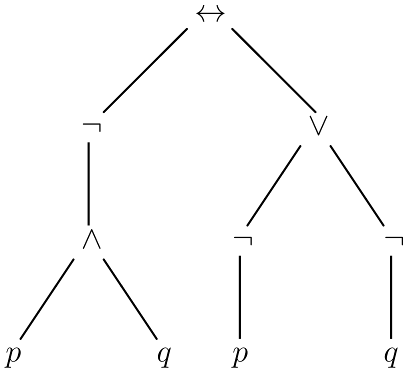
Example: decomposition tree of a logic formula

We can represent complicated expressions, like formulas of propositional logic and arithmetical expressions, using rooted trees.

FOR EXAMPLE: The formula

$$\big(\neg(p \wedge q) \leftrightarrow (\neg p \vee \neg q)\big)$$

can be represented as



Binary search trees: a tool for sorting linearly ordered lists

Linearly ordered list: a sequence (list) whose elements are linearly ordered (not necessarily in the order of listing)

FOR EXAMPLE:

- $(5, 128, 3, 2, 15, 4, 20)$ is a list of natural numbers, natural numbers can be ordered by the $\leq$ relation (which is a linear order, see lecture slide 92)
- (*mathematics*, *physics*, *geography*, *geology*, *psychology*) is a list of words, words can be ordered by the *lexicographical order relation* $\prec$ (see lecture slide 203)

Searching for items in a *linearly ordered list* is an important task. Binary search trees are particularly useful in representing elements in such a list. There are very efficient methods for

- *searching* data in binary search trees,
- *revising* data in binary search trees,
- converting linearly ordered lists to binary search trees and back.

Example linear order: lexicographical order on words

First, we order the letters of the English alphabet as usual:

$$a \prec b \prec c \prec d \prec e \prec \ldots \prec x \prec y \prec z$$

Then, we can use this ordering of the letters to order longer words:

- Given two words w_1 and w_2, we compare them letter by letter, from left to right, passing equal letters.
- If at any point a letter in w_1 is $\prec$-smaller than the corresponding letter in w_2, then we put $w_1 \prec w_2$.
- If every letter in w_1 is equal to the corresponding letter in w_2, but w_2 is longer than w_1, then we also put $w_1 \prec w_2$.
- In any other case, we put $w_2 \prec w_1$.

FOR EXAMPLE: *discreet* $\prec$ *discreetness* $\prec$ *discrete* $\prec$ *discretion*

geography $\prec$ *geology* $\prec$ *mathematics* $\prec$ *physics* $\prec$ *psychology*

Binary search trees

We are given two things: a list $\boxed{L}$ of items, and a linear order $\boxed{\prec}$ on them.

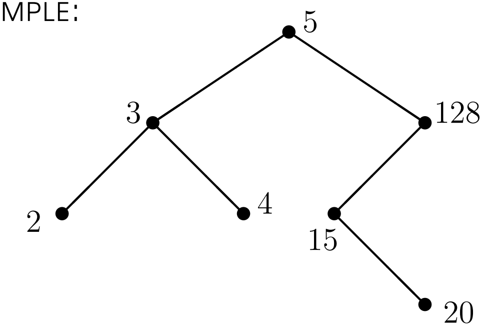
A **binary search tree** for L and $\prec$ is a binary tree in which every vertex is labelled with an item from L such that:

(1) the label of each vertex

- is $\prec$-greater than the labels of all vertices in its left subtree,
- is $\prec$-less than the labels of all vertices in its right subtree.

(2) Also, every path in the tree is 'compatible with' the order of listing.

FOR EXAMPLE:



for the list $(5, 128, 3, 2, 15, 4, 20)$ and linear order $\leq$

How to build binary search trees from linearly ordered lists

We are given a list L of items, and a linear order $\prec$ on them.
We go through each member of the list, from left to right:

- **First item:** We assign it as the label of the root.
- **Comparing:** We take the next item on the list, and first we compare it with the labels of the 'old' vertices already in the tree, starting from the root and
  - moving to the left if the new item is $\prec$-less than the label of the respective 'old' vertex, if this 'old' vertex has a left child, or
  - moving to the right if the new item is $\prec$-greater than the label of the respective 'old' vertex, if this 'old' vertex has a right child.
- **Adding:**
  - When the new item is $\prec$-less than the label of an 'old' vertex and the vertex has no left child, then we insert a new left child to the 'old' vertex, and label it with the new item.
  - When the new item is $\prec$-larger than the label of an 'old' vertex and the vertex has no right child, then we insert a new right child to the 'old' vertex, and label it with the new item.

Building a binary search tree: an example

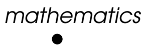
TASK: Build a binary search tree for the list of words

↓

mathematics, physics, geography, zoology, meteorology, geology, psychology,

using lexicographical order $\boxed{\prec}$.

1ST STEP: We take *mathematics* and label the root with it:



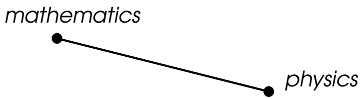
↓

mathematics, physics, geography, zoology, meteorology, geology, psychology

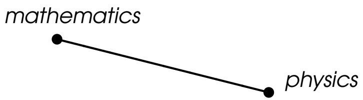
2ND STEP: We take *physics* and compare it with *mathematics*:

$$mathematics \prec physics\,,$$

mathematics has no right child, so we label a new right child with *physics*:



Building a binary search tree: an example (cont.)



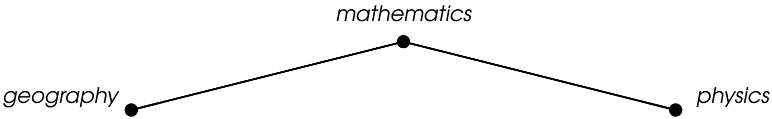
↓

mathematics, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*

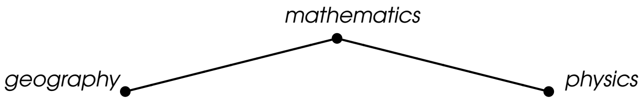
3RD STEP: We take *geography* and compare it with *mathematics*:

$$\textit{geography} \prec \textit{mathematics},$$

mathematics has no left child, so we label a new left child with *geography*:



Building a binary search tree: an example (cont.)



↓

mathematics, physics, geography, zoology, meteorology, geology, psychology

4TH STEP: We take *zoology* and compare it with *mathematics*:

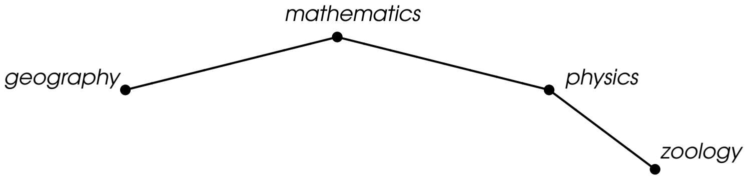
$$mathematics \prec zoology,$$

so we move to the right child of the root and take its label, *physics*.

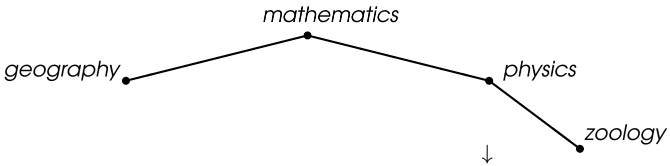
5TH STEP: We compare the new word *zoology* with *physics*:

$$physics \prec zoology,$$

physics has no right child, so we label a new right child with *zoology*:



Building a binary search tree: an example (cont.)



mathematics, physics, geography, zoology, meteorology, geology, psychology

6TH STEP: We take *meteorology* and compare it with *mathematics*:

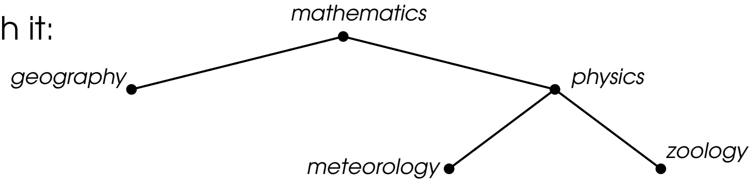
$$\textit{mathematics} \prec \textit{meteorology}\,,$$

so we move to the right child of the root and take its label, *physics*.

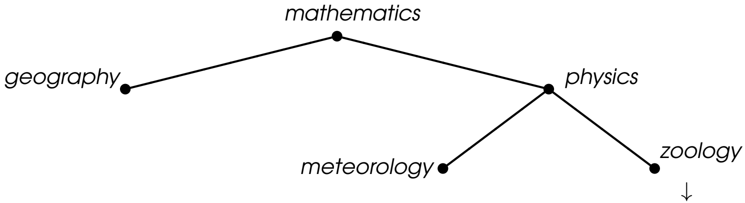
7TH STEP: We compare the new word *meteorology* with *physics*:

$$\textit{meteorology} \prec \textit{physics}\,,$$

so we label a new left child with it:



Building a binary search tree: an example (cont.)



mathematics, physics, geography, zoology, meteorology, geology, psychology

8TH STEP: We take *geology* and compare it with *mathematics*:

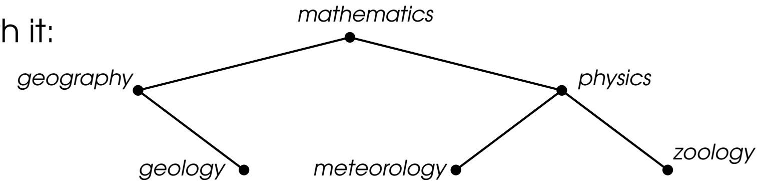
$$geology \prec mathematics\,,$$

so we move to the left child of the root and take its label, *geography*.

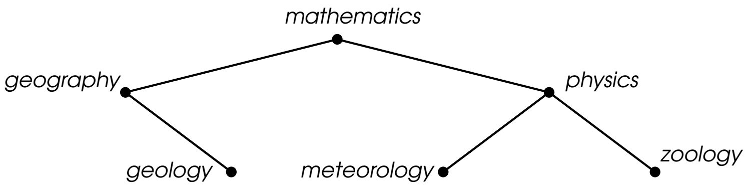
9TH STEP: We compare the new word *geology* with *geography*:

$$geography \prec geology\,,$$

so we label a new right child with it:



Building a binary search tree: an example (cont.)



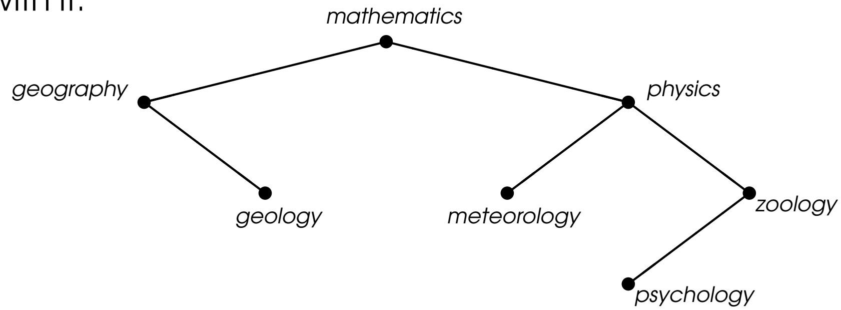
↓

mathematics, physics, geography, zoology, meteorology, geology, psychology

FINALLY: We take *psychology*, then compare it with *mathematics*, move to the right, compare it with *physics*, move to the right, then compare it with *zoology*. As

$$\textit{psychology} \prec \textit{zoology},$$

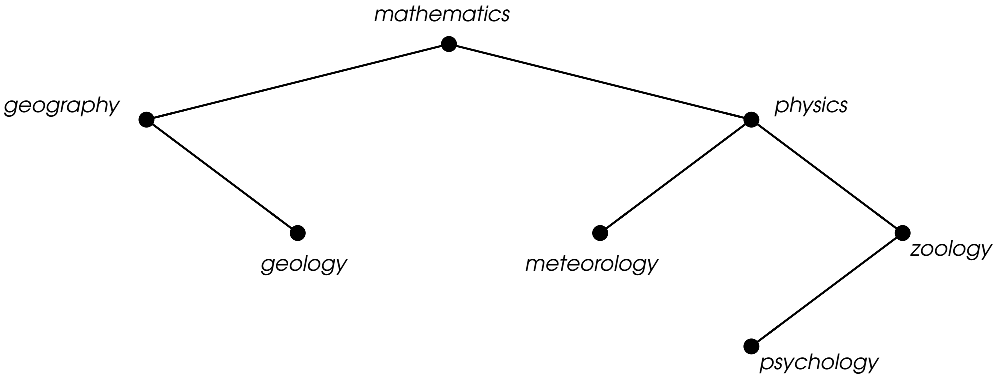
we label a <u>new left child</u> with it:



Binary search trees: locating or adding items I

TYPICAL TASK: We already have a binary search tree. We are given a word, *meteorology*. How many comparisons do we need to locate this word in our tree (if it is there), or to add to it (if it is new)?

SOLUTION: Just 3. We take the word. First compare it with *mathematics*, move to the right, then compare it with *physics*, move to the left, then compare it with *meteorology*: successfully located.



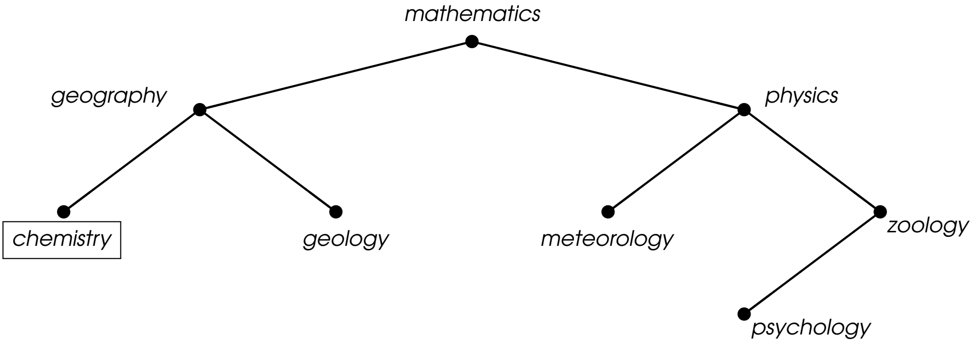
Binary search trees: locating or adding items II

TASK: We are given a word, *chemistry*. How many comparisons do we need to locate or add it?

SOLUTION: Just 2. We compare it with *mathematics*, move to the left, then compare it with *geography*. As

$$\textit{chemistry} \prec \textit{geography},$$

and *geography* has no left child, we know at this point that *chemistry* is NOT in the tree. So we create a new left child and label it with *chemistry*:



Tree traversal

Rooted trees are often used to store information. We need systematic procedures for visiting each vertex of a rooted tree to access data. Such procedures are called *traversal algorithms.*

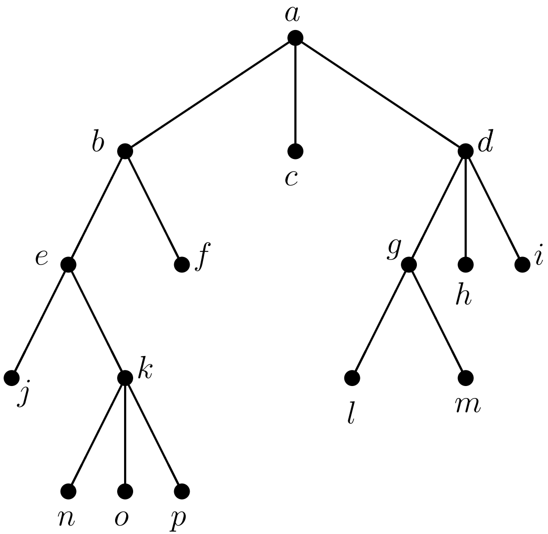
Here are some important ones:

- **Preorder traversal:** Visit the root, then continue traversing subtrees in preorder, from left to right.

- **Inorder traversal:** Begin traversing leftmost subtree in inorder, then visit root, then continue traversing subtrees in inorder, from left to right.

- **Postorder traversal:** Begin traversing leftmost subtree in postorder, then continue traversing subtrees in postorder, from left to right, finally visit root.

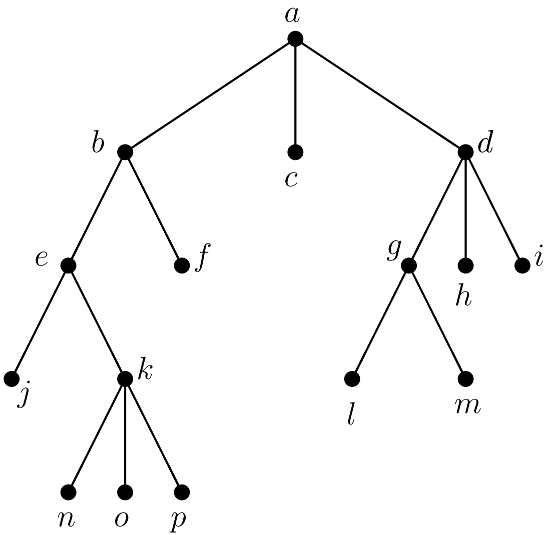
Preorder traversal



Visit the root, then continue traversing subtrees in preorder, from left to right:

$$a, \quad b, \quad e, \quad j, \quad k, \quad n, \quad o, \quad p, \quad f, \quad c, \quad d, \quad g, \quad l, \quad m, \quad h, \quad i$$

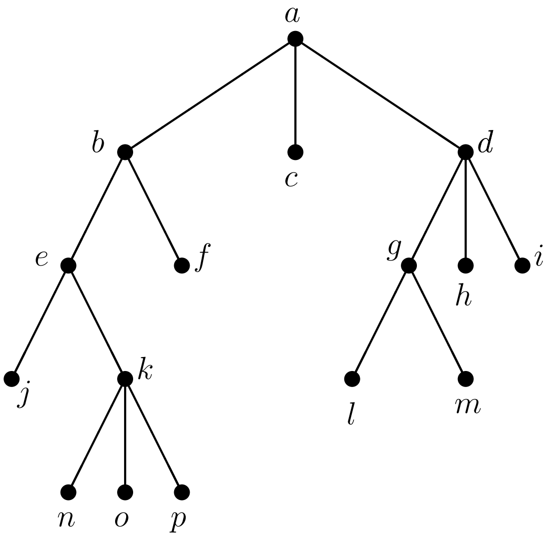
Inorder traversal



Begin traversing leftmost subtree in inorder, then visit root, then continue traversing subtrees in inorder, from left to right:

$$j,\quad e,\quad n,\quad k,\quad o,\quad p,\quad b,\quad f,\quad a,\quad c,\quad l,\quad g,\quad m,\quad d,\quad h,\quad i$$

Postorder traversal



Begin traversing leftmost subtree in postorder, then continue traversing subtrees in postorder, from left to right, finally visit root:

$$j,\quad n,\quad o,\quad p,\quad k,\quad e,\quad f,\quad b,\quad c,\quad l,\quad m,\quad g,\quad h,\quad i,\quad d,\quad a$$

Finite automata: where they are used

Finite automata (aka **finite-state machines**) are extensively used in computer science and data networking applications.

Here are some examples of the areas where they are used:

- programs for spell checking
- pattern matching in search engines
- compiler design
- specifying network protocols
- chip design
- speech recognition
- transforming text using markup languages like XML
- software optimisation
- . . .

Preliminaries: alphabets and words

- An **alphabet** is a finite set S of symbols.

  FOR EXAMPLE:
  - $\{a, b, c, \ldots, z\}$
  - $\{0, 1\}$
  - $\{\Box, \Diamond, \heartsuit\}$

- A **word** or **string** (over an alphabet S) is a finite sequence of symbols from S, written without commas.

  FOR EXAMPLE:
  - $tortoise$, $azwzax$
  - 1111111111000000000, 000110,
  - $\heartsuit\heartsuit\Box$, $\Box\Diamond\Box\Box\Diamond\heartsuit$,
  - the **empty word**, denoted by $\boxed{\varepsilon}$, is a word over *any* alphabet (but we may assume that ε is *NOT a symbol* of any of our alphabets)

More on words

- The **length** of a word w is

$$|w| = \text{the number of symbols in } w$$

FOR EXAMPLE: $|azwza| = 5,\qquad |\heartsuit\heartsuit\Box| = 3,\qquad |\varepsilon| = 0$

- The **concatenation** of words x and y (written xy):

  the word x followed by the word y

  - $w^n = \overbrace{ww\dots w}^{n}$

    FOR EXAMPLE: $(\heartsuit\Box)^0 = \varepsilon,\qquad (01)^3 = 010101,\qquad a^4 = aaaa$

  - $x\varepsilon = \varepsilon x = x$, for every word x

- If $w = xy$ then x is a **prefix** of w, and y a **suffix** of w.

  FOR EXAMPLE: tor is a prefix and se is a suffix of $tortoise$

  $tortoise$ is **both** a prefix and a suffix of $tortoise$

Preliminaries: languages

A **language** (over an alphabet S) is a set of words over S.

FOR EXAMPLE:

(1) $S = \{a, b, c, \ldots, z\}$

- L_1 = all English words
- L_2 = all Latin words
- $L_3 = \{kdpekvg,\ leih,\ hkiiw,\ wowiszk\}$

(2) $S = \{0, 1\}$

- $L_4 = \{001,\ 101010,\ 111,\ 1001\}$
- $L_5 = \{0^n 1^m \mid n$ is an even, m is an odd number$\}$

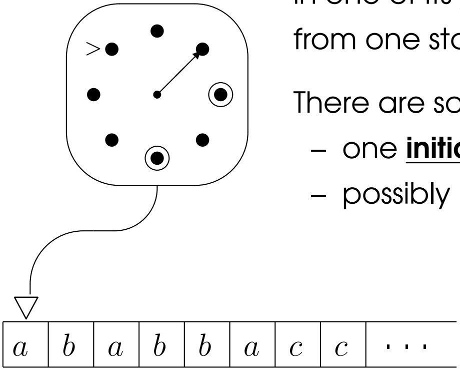
Finite automaton

A *theoretical model* for programs using a constant amount of memory regardless of the input form.

Finite control device: In any moment it can be in one of its **states**. It is hard-wired how it changes from one state to another.

There are some special states:

- one **initial state**,
- possibly more **favourable states**.



Reading head

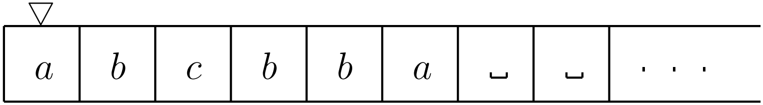
Input tape: It is divided into cells, having a leftmost cell, but as long as we want to the right. Each cell may contain one character of the **input alphabet**.

Finite automaton: how it starts

- The finite control device is in its unique initial state.

- The tape contains a finite word of the input alphabet, the **input**, at its left end. The remainder of the tape contains only blank cells.

- The reading head is positioned on the leftmost cell of the input tape, containing the first character of the input word.



Finite automaton: how it works

- At regular time intervals, the automaton
  - reads one character from the input tape,
  - moves the reading head one cell to the right, and
  - chooses the next state of its control device.

- The control device is hard-wired such that the next state depends
  on (1) the previous state, and
  on (2) the character read from the tape.

- As the input is finite, at some moment the reading head reaches the end of the input word (that is, the first blank cell).

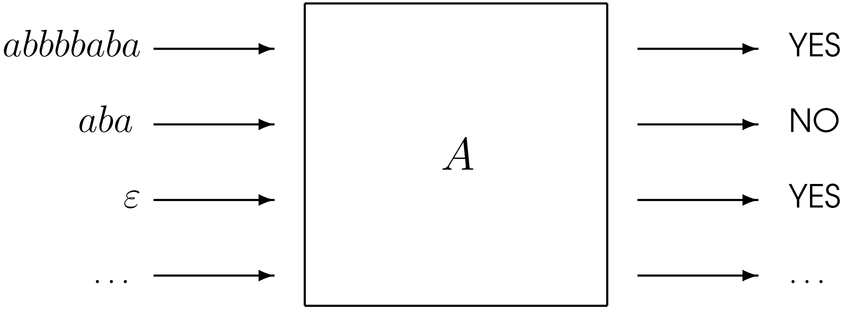
  If at this moment the control device is in a favourable state,
  then the input word is **accepted** by the automaton.

  Otherwise, the input word is not accepted (**rejected**).

Finite automata: what they are used for

- Each finite automaton is a kind of 'recognition' or 'decision' device over all possible words of its input alphabet.

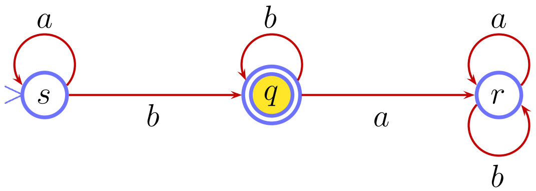
- Each automaton can be 'tried' on infinitely many input words, and gives a YES/NO answer each time.



‘Visual’ representation of finite automata: state transition diagrams

We can represent the hard-wired control device of a finite automaton by a directed multigraph:

- with the vertices representing the states,
- and the arrow-edges being labelled by symbols of the input alphabet.

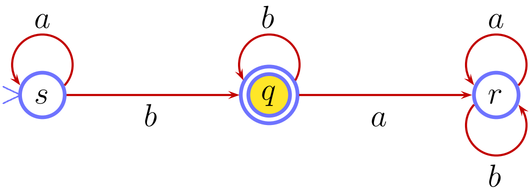


- The initial state is marked by $>$.
- The favourable states are double-circled.
- Each arrow represents a possible **transition**, hard-wired in the control device:

  Say, an arrow from state q to state r labelled by symbol a indicates that, when the head is reading a and the control device is in state q, then it should move next to state r.

Finite automata: an example

Automaton A_1 :



- <u>Input 1</u>: $abba$

  <u>Computation</u>: $(s, abba),\ (s, bba),\ (q, ba),\ (q, a),\ (r, \varepsilon)$

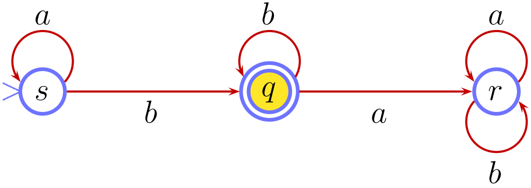
  $\rightsquigarrow$ word $abba$ is **rejected**

- <u>Input 2</u>: $aabb$

  <u>Computation</u>: $(s, aabb),\ (s, abb),\ (s, bb),\ (q, b),\ (q, \varepsilon)$

  $\rightsquigarrow$ word $aabb$ is **accepted**

More on how automaton A_1 works



- Input 3: $bbaaa$

  Computation: $(s, bbaaa),\ (q, baaa),\ (q, aaa),\ (r, aa),\ (r, a),\ (r, \varepsilon)$

  $\rightsquigarrow$ word $bbaaa$ is **rejected**

- Input 4: bbb

  Computation: $(s, bbb),\ (q, bb),\ (q, b),\ (q, \varepsilon)$

  $\rightsquigarrow$ word bbb is **accepted**

- Input 5: ε

  Computation: (s, ε)

  $\rightsquigarrow$ word ε is **rejected**

Deterministic Finite Automaton (DFA): a summary

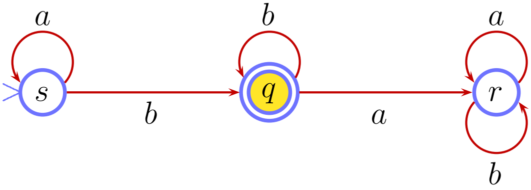
In order to describe a $\boxed{\textbf{DFA}}$ we need to describe **5** things:

- its **states**
- its **input alphabet**,
- its (unique) **initial state**,
- its **favourable** (or **accepting**) **states** (there can be none, or more than one)
- its **transition function**: for every (state, input symbol) pair we have to tell what the next state should be

A word w is **accepted** by DFA A if the computation of A on input w ends up in some *favourable state.*

Otherwise, w is **rejected** by A.

DFAs: what does ‘deterministic’ mean?



Observe that for each state and each symbol, there is a *unique* arrow coming out of the state labelled by the symbol.

(Here *unique* means two things: there is one, but not more than one.)

In other words, the pair

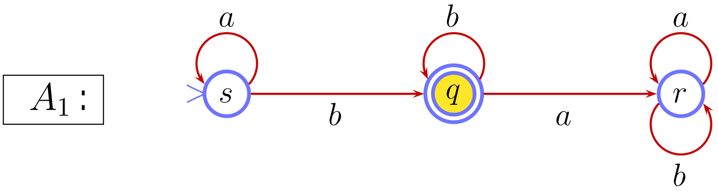
(current state, symbol read)

uniquely determines the next state.

This is why **D**FAs are called **deterministic**.

(Later we will also consider automata that are NOT like this.)

Another representation of DFAs: the transition table



states: s, q, r input alphabet: $\{a, b\}$ initial state: s favourable states: q

The **transition table** is another way of representing the transition function of A_1:

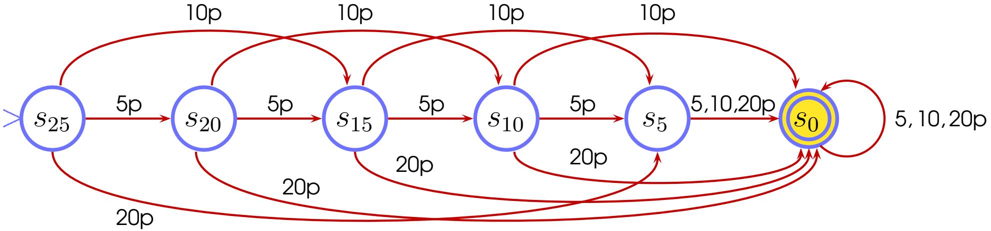
	a	b
s	s	q
q	r	q
r	r	r

Observe again: for each 'cell' in the transition table there is a *unique* state to put in. In other words, the pair

(current state, symbol read)

uniquely determines the next state. (This is why DFAs are called **deterministic**.)

Example A_2: vending machine



states: $s_{25}, s_{20}, s_{15}, s_{10}, s_5, s_0$

input alphabet: $\{5\text{p},\ 10\text{p},\ 20\text{p}\}$

initial state: s_{25}

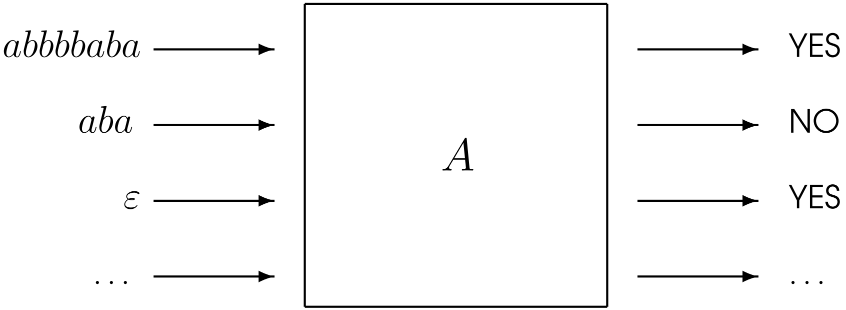
favourable states: s_0.

Word: any sequence of 5p, 10p and 20p coins

Accepts: all the words such that the sum of the coins is $\geq$ 25p

Languages and DFAs

RECALL: A **language** (over an alphabet S) is a set of words over S.



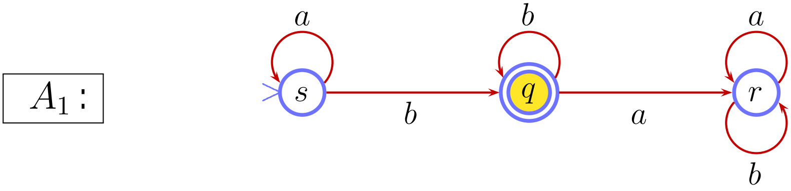
Any DFA may accept certain words, while may reject others.

If we collect all words accepted by a DFA A, we obtain a language:

the **language of a DFA** A is

$L(A) =$ the set of all the words over its input alphabet that A **accepts**

DFA A_1 revisited



We already know:

- accepted: $aabb$, bbb
- rejected: $abba$, $bbaaa$, ε

$$
\begin{aligned}
L(A_1) &= \text{all the words starting with some (possibly none) } a\text{s} \\
&\qquad\qquad \text{followed by at least one } b \\
&= \{a^n b^m \mid n = 0, 1, 2, \ldots,\ \ m = 1, 2, \ldots\}
\end{aligned}
$$

How to find the language of a finite automaton

(1) Experiment with words.

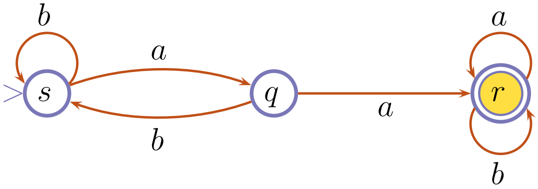
(2) Come up with a language.

(3) Test the suggested language:

- Every word that is accepted by the automaton should be in the suggested language.
- Every word that is rejected by the automaton must not be in the suggested language.

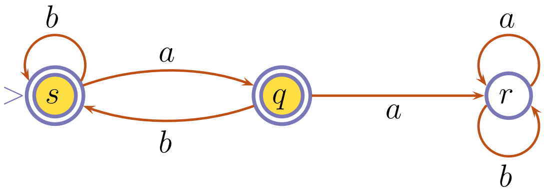
(4) Revise the suggested language if necessary, then go to step (3).

Example: DFA A_3



$L(A_3)$ = all the words that contain two consecutive a s

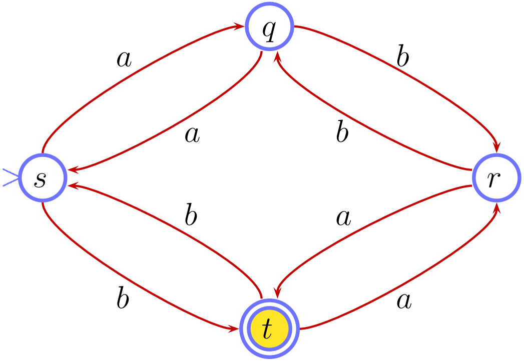
Example: DFA A_4



$L(A_4)$ = all the words that do not contain two consecutive a s

Observe that $L(A_4) = \{w \mid w$ is any word of a s and b s$\} - L(A_3)$.

A trickier example: DFA A_5



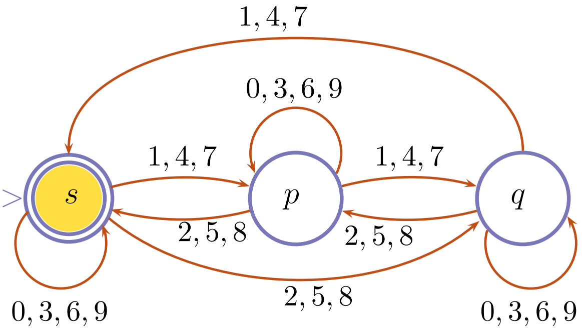
SOME EXPERIMENTS:

- $abbabbabaa$ — ends up in r $\rightsquigarrow$ rejected
- $abbaabbabaa$ — ends up in t $\rightsquigarrow$ accepted
- $abbababbabaa$ — ends up in s $\rightsquigarrow$ rejected
- $babbababbabaa$ — ends up in t $\rightsquigarrow$ accepted
- $babbababbabba$ — ends up in q $\rightsquigarrow$ rejected

$L(A_5) =$ all the words that contain an even number of a s and odd number of b s

Example: DFA A_6

Input alphabet: $0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9$



	0	1	2	3	4	5	6	7	8	9
s	s	p	q	s	p	q	s	p	q	s
p	p	q	s	p	q	s	p	q	s	p
q	q	s	p	q	s	p	q	s	p	q

$$\boxed{L(A_6) = \ ?}$$

DFAs vs. languages

- So far we discussed how to determine the language of a DFA.

  $$A \quad \leadsto \quad L(A)$$

  program $\leadsto$ specification

- There is the 'reverse' task:

  Given a language L, design a DFA A such that the language $L(A)$ of A is L.

  $$L \quad \leadsto \quad A \ \text{ with } L(A) = L$$

  specification $\leadsto$ program

Designing deterministic finite automata

Task: *Design a DFA* A *such that* $L(A)$ *consists of*

all *the strings of* 0*s and* 1*s ending with* 11.

How? creative task, needs thinking

Slogan: states = memory

(1) Determine what to remember about the input string,
while the head is reading through it, **from left to right.**

Assign a state to each of the possibilities:

- state "waiting for 11"
- state "waiting for 1"
- state "OK at the moment"

(2) Add the transitions telling how the possibilities rearrange, and select the initial state and the favourable states.

(3) Test the automaton.

Designing deterministic finite automata (cont.)

It is sufficient to remember three ‘states’ of the **string read so far:**

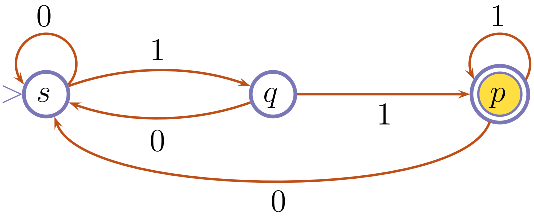
- The string does not end in 1. (Either it is ε or it ends in 0.)

  $\leadsto$ initial state s (“waiting for 11”)

- The string is 1 or ends in 01. $\leadsto$ state q (“waiting for 1”)

- The string ends in 11.

  $\leadsto$ favourable state p (“OK at the moment”)



	0	1
s	s	q
q	s	p
p	s	p

Designing deterministic finite automata II

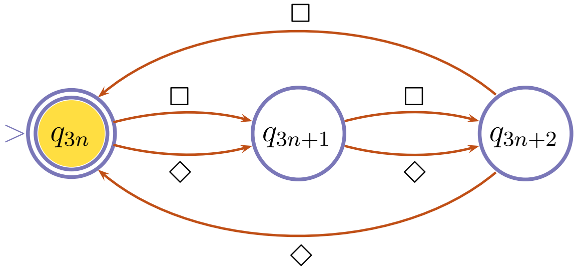
Task: *Design a DFA A such that $L(A)$ consists of* **all** *the strings of $\Box$s and $\Diamond$s whose length is divisible by 3 (that is, either 0, or 3, or 6, or $\dots$, or $6033, \dots$)*

(1) Determine what to remember about the input string, while the head is reading through it, from left to right.

Assign a state to each of the possibilities: q_{3n}, q_{3n+1}, q_{3n+2}.

(2) Add the transitions telling how the possibilities rearrange, and select the initial state and the favourable states.

(3) Test the automaton.



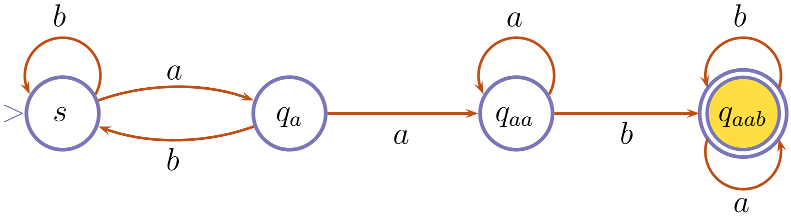
	□	◇
q_{3n}	q_{3n+1}	q_{3n+1}
q_{3n+1}	q_{3n+2}	q_{3n+2}
q_{3n+2}	q_{3n}	q_{3n}

Pattern matching

Task: *Design a DFA* A *such that* $L(A)$ *consists of* **all** *the strings of* a*s and* b*s that contain* $\boxed{aab}$ *as a substring.*

There are four possibilities to remember:

- 'haven't seen any symbols of the pattern:' (initial) state s
- 'have just seen an a:' state q_a
- 'have just seen aa:' state q_{aa}
- 'have just seen the entire pattern aab:' (favourable) state q_{aab}



	a	b
s	q_a	s
q_a	q_{aa}	s
q_{aa}	q_{aa}	q_{aab}
q_{aab}	q_{aab}	q_{aab}

DFA as a theoretical model for programming

- – unusual (visual), different from conventional programming languages
- + major concepts of imperative programming (e.g. sequences, branching, loops) can be simulated
- + elegant, simple to use (visual), has a thoroughly developed theory
- + all pattern matching (=first step in WWW search engines) can be described
- – limited expressive power (e.g. only constant amount of memory is available)

What about ‘method-calling’?

Combining two automata

Task: *Design an automaton A such that $L(A)$ consists of* **all** *the strings of as and bs that have either two consecutive as or two consecutive bs.*

- On lecture slide 236 there is a DFA A_3 accepting all the strings that have two consecutive as.

  This automaton can easily be transformed into an automaton A_3' that accepts all the strings that have two consecutive bs.

- Since we already have the 'methods' that solve the problem for aas and, respectively, bbs, an obvious idea would be to design a 'main program' that would just 'call' the given methods.

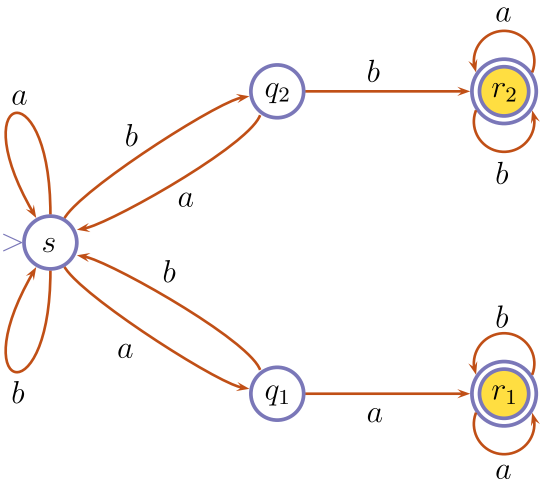
  A possible solution would be to 'combine' the initial states of A_3 and A_3' (see next lecture slide).

Combining two automata (cont.)

Given: $L(A_3) =$ all the words having two consecutive as

$L(A_3') =$ all the words having two consecutive bs

Needed: automaton A such that $L(A) = L(A_3) \cup L(A_3')$



Nondeterminism

Observe that the state transition diagram on the previous slide no longer represents a **DFA:**

- there is more than one a-arrow leaving state s,
- there is more than one b-arrow leaving state s.

We say that this automaton is **nondeterministic:**

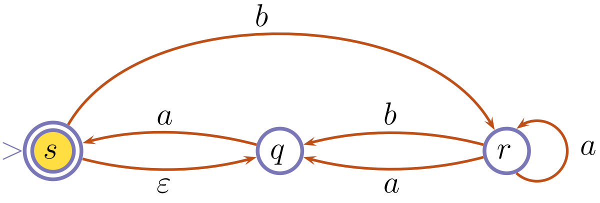
Because, say, being in state s and reading a from the input tape,
the device now has a choice: it can choose to move **either** to state q_1
or to state s.
In other words, the next state is *not determined*
by the previous state and the symbol read.

But then what does it mean that a word is accepted??

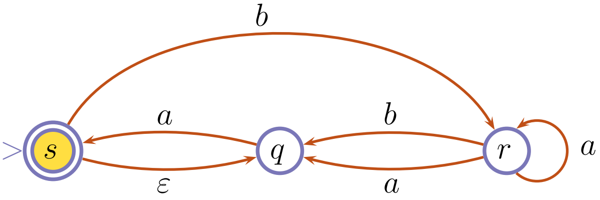
More nondeterminism

Apart from having a choice in certain situations,
there are other permitted features in nondeterministic finite automata (**NFA**):

- There can be states and symbols such that **no** arrow labelled by that symbol leaves the state in question: reading that symbol in that state the device gets stuck (say, in the example below, there is no a-arrow leaving s).
- We also allow the machine to '**jump**' from a state to another state, without 'consuming' any input symbol from the tape (that is, the head does not move when 'jump' happens). The allowed 'state-jumps' will be indicated on the state transition diagrams by arrows labelled by the empty word ε.



Representing NFAs by transition tables



states: s, q, r input alphabet: $\{a, b\}$ initial state: s favourable states: s

transition table:

	a	b	ε
s	–	r	q
q	s	–	–
r	$q,\ r$	q	–

Observe the differences from DFAs:

- there can be 'cells' in the table containing no or more than one states
- the table has an ε-column (for possible jumps)

An NFA can be regarded as a DFA if all cells in the ε-column are empty, and in every other column every cell contains a single state.

Acceptance of words by NFAs

In an NFA, there can be *more than one* computation on an input word.

Some of these may end up in a favourable state, while some others either get stuck, or may end up in a non-favourable state.

So, what is it supposed to mean that a word is **accepted** by an NFA ?

A word w is **accepted** by NFA A if *there is* a computation of A on input w that ends up in some *favourable state.*

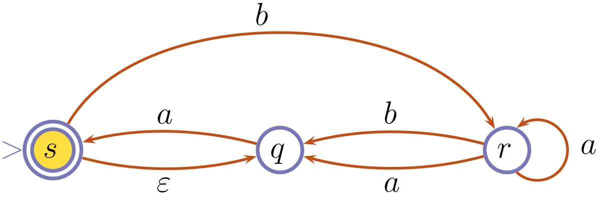
- watch out: there can be many other, 'bad' computations on w
- watch out: 'getting stuck' in a favourable state does not mean acceptance (ends up = all symbols of the input word have been read)

Otherwise, A **rejects** the input word.

So an NFA can reject an input word w because *every* computation on w

- is either 'stuck'
- or ends up in a non-favourable state

Example: how an NFA works



	a	b	ε
s	−	r	q
q	s	−	−
r	$q,\ r$	q	−

- Computations on input $baba$:

$(s, baba), (q, baba)$ (stuck)

$(s, baba), (r, aba), (r, ba), (q, a), (s, \varepsilon)$

$\dots$

$\leadsto$ $baba$ is $\boxed{\text{accepted}}$

- Computations on input aa:

$(s, aa), (q, aa), (s, a), (q, a), (s, \varepsilon)$ $\}$ $\leadsto$ aa is $\boxed{\text{accepted}}$

- Computations on input $babba$:

$(s, babba), (q, babba)$ (stuck)

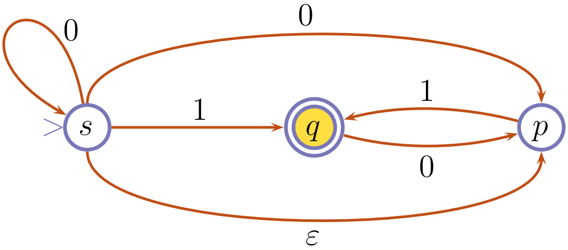
$(s, babba), (r, abba), (r, bba), (q, ba)$ (stuck)

$(s, babba), (r, abba), (q, bba)$ (stuck, no more)

$\leadsto$ $babba$ is $\boxed{\text{rejected}}$

NFA: another example

	0	1	ε
s	$s,\ p$	q	p
q	p	–	–
p	–	q	–



- Computations on input 11:

$(s, 11), (q, 1)$ (stuck)
$(s, 11), (p, 11), (q, 1)$ (stuck, no more)
$\}$ $\rightsquigarrow$ 11 is $\boxed{\text{rejected}}$

- Computations on input 00:

$(s, 00), (s, 0), (s, \varepsilon)$
$(s, 00), (s, 0), (s, \varepsilon), (p, \varepsilon)$
$(s, 00), (s, 0), (p, \varepsilon)$
$(s, 00), (s, 0), (p, 0)$ (stuck)
$(s, 00), (p, 0)$ (stuck)
$(s, 00), (p, 00)$ (stuck, no more)
$\}$ $\rightsquigarrow$ 00 is $\boxed{\text{rejected}}$

- Computations on input 001:

$(s, 001), (s, 01), (s, 1), (q, \varepsilon)$
$\ldots$
$\}$ $\rightsquigarrow$ 001 is $\boxed{\text{accepted}}$

NFAs: what the whole fuss is about?

- Nondeterminism can be viewed as a kind of parallel computation model wherein several ‘processes’ can be running concurrently (‘threads’).

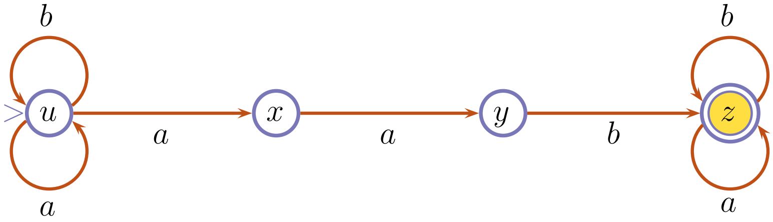
- We will see:
  Nondeterministic finite automata are much **easier to design** than deterministic ones.

- We will also see:
  Whatever can be done by a nondeterministic finite automaton, it can also be done (though usually in a bit more complicated way) by a deterministic one. In other words, nondeterminism **does not increase the computational power** of finite automata.

- NFAs are used in hardware (chip) design:
  - Given a task, an NFA is designed.
  - Then this NFA is turned to a DFA doing the same thing.
  - Then the resulting DFA is ‘minimised’.
  - The obtained minimal DFA is hard-wired.

Pattern matching revisited

Task: *Design a finite automaton* A *such that* $L(A)$ *consists of* **all** *the strings of* a*s and* b*s that contain* $\boxed{aab}$ *as a substring.*



	a	b	ε
u	$u,\ x$	u	–
x	y	–	–
y	–	z	–
z	z	z	–

- Given the same task, it is much easier to design an NFA than a DFA for it: We don't have to worry about <u>all</u> computations. We just need to ensure that
  - if a word should be accepted,
    then <u>there is at least one 'good' computation</u> for it
  - if a word should be rejected, then there is <u>no 'good' computation</u> for it

Equivalence of DFAs and NFAs

- And it really does not matter:
  We will now see that every NFA can be turned to a DFA
  accepting precisely the same words.

Two automata A and A' that accept the same language are said to be **equivalent**.

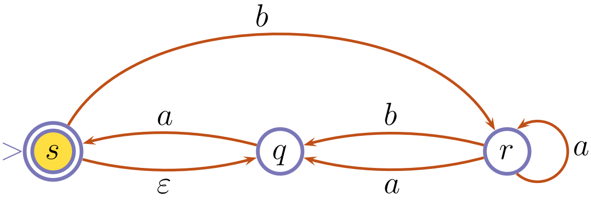
$$\boxed{\textbf{For every NFA } A\textbf{, there is a DFA } A' \textbf{ equivalent to } A\textbf{.}}$$

Moreover, there is a 'mechanical method' (an *algorithm*) that carries out this conversion from a nondeterministic automaton to its deterministic equivalent:

$\leadsto$ the 'Subset Construction'

The Subset Construction: turning NFAs to equivalent DFAs

Task: Given an NFA



turn it to an equivalent **D**FA (**deterministic !**)

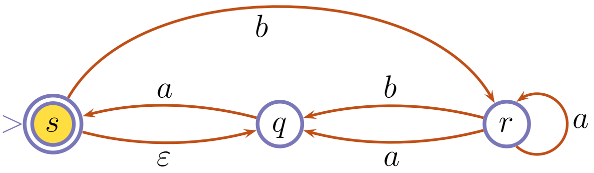
- watch out: in a DFA **no** choice, **no** 'getting stuck', **no** ε-jumps are allowed

Things to define:

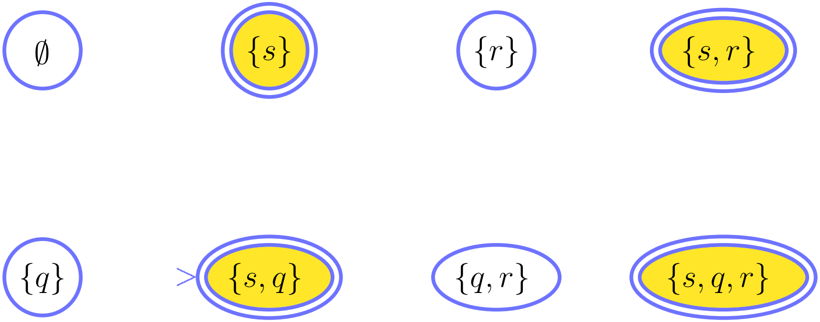
- new states $\boxed{?}$
- new initial state $\boxed{?}$
- new favourable states $\boxed{?}$
- new transitions (arrows) $\boxed{?}$

Subset construction: the new states

NFA:



- new states: named after **subsets** of the original NFA's states
- new initial state: the **set** containing the original NFA's initial state, **plus** all states that are reachable from it by some ε-jumps
- new favourable states: those **subsets** that contain **at least one** of the original NFA's favourable states

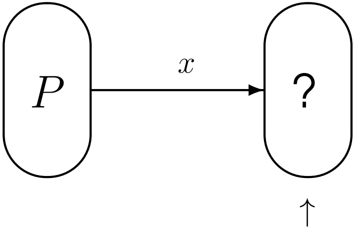


Subset construction: the new transition function

RECALL: Being in a state and reading an input symbol, the transition function tells the automaton's next state.

And the new states are named after *subsets* of the 'old' states now.

General rule:



for each symbol x in the input alphabet

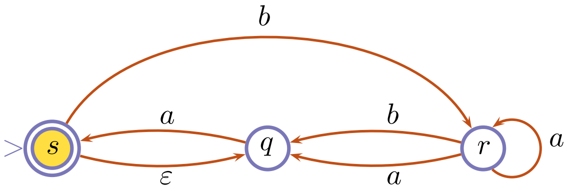
the set of all those states that are reachable from some state in the set P

- either by an x-arrow,
- or by an x-arrow followed by possibly one or more ε-jumps

('first ε-jump(s) then x-arrow' – no good)

Example: computing the new transition function

NFA:



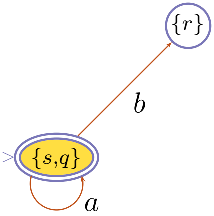
In a DFA **every** state **must have** one a-arrow out and one b-arrow out.

We begin with the new initial state, and compute where these two arrows go from it:

Transition table format:

	a	b
$\{s,q\}$	$\{s,q\}$	$\{r\}$

State transition diagram format:



WHY? Let's apply the general rule of lecture slide 259:

- from s we cannot reach anything by an a-arrow
- from q we can reach $\boxed{s}$ by an a-arrow
- from q we can reach $\boxed{q}$ by an a-arrow followed by an ε-jump

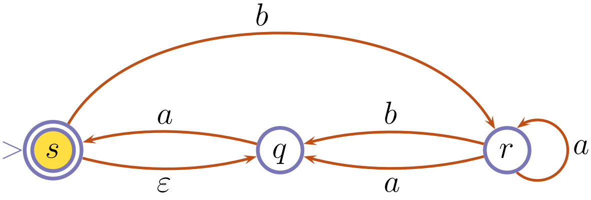
So altogether, the a-arrow from state $\{s,q\}$ goes to state $\{s,q\}$ (itself).

- from s we can reach $\boxed{r}$ by a b-arrow
- from q we cannot reach anything by a b-arrow

So altogether, the b-arrow from state $\{s,q\}$ goes to state $\{r\}$.

Example: computing the new transition function (cont.)

NFA:

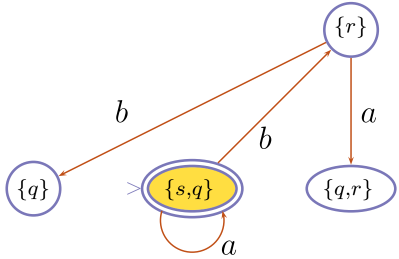


Next, for every obtained new state, we compute where the a-arrow and the b-arrow goes from it. We do this until there are no newly obtained states.

Transition table format:

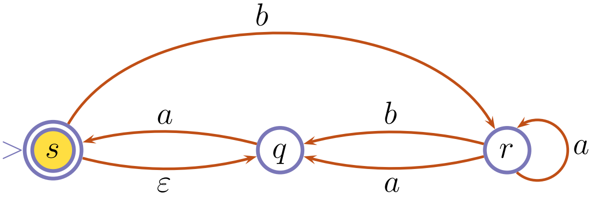
	a	b
$\{s,q\}$	$\{s,q\}$	$\{r\}$
$\{r\}$	$\{q,r\}$	$\{q\}$

State transition diagram format:



Example: computing the new transition function (cont.)

NFA:

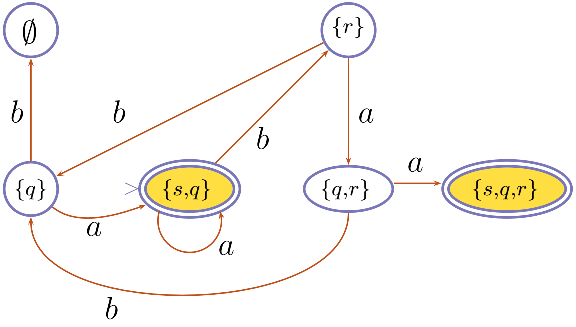


Next, for every obtained new state, we compute where the a-arrow and the b-arrow goes from it. We do this until there are no newly obtained states.

Transition table format:

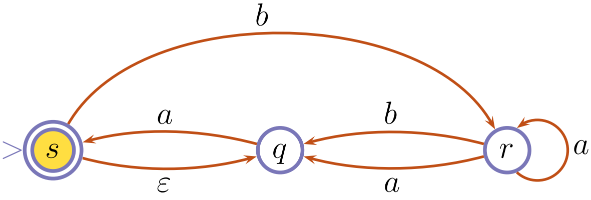
	a	b
$\{s,q\}$	$\{s,q\}$	$\{r\}$
$\{r\}$	$\{q,r\}$	$\{q\}$
$\{q,r\}$	$\{s,q,r\}$	$\{q\}$
$\{q\}$	$\{s,q\}$	$\emptyset$

State transition diagram format:



Example: computing the new transition function (last step)

NFA:

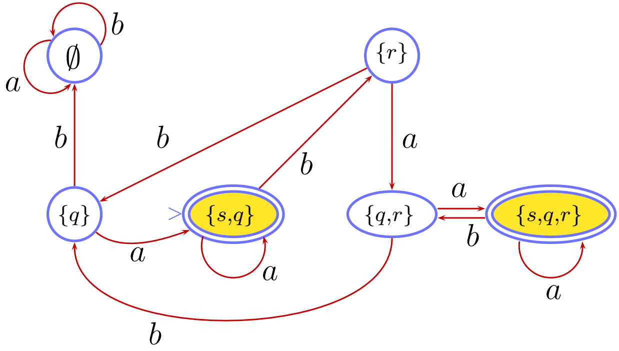


Next, for every obtained new state, we compute where the a-arrow and the b-arrow goes from it. We do this **until there are no newly obtained states.**

Transition table format:

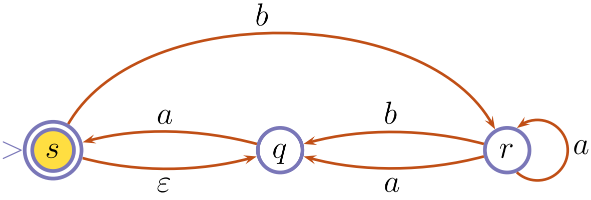
	a	b
$\{s,q\}$	$\{s,q\}$	$\{r\}$
$\{r\}$	$\{q,r\}$	$\{q\}$
$\{q,r\}$	$\{s,q,r\}$	$\{q\}$
$\{q\}$	$\{s,q\}$	$\emptyset$
$\{s,q,r\}$	$\{s,q,r\}$	$\{q,r\}$
$\emptyset$	$\emptyset$	$\emptyset$

State transition diagram format:

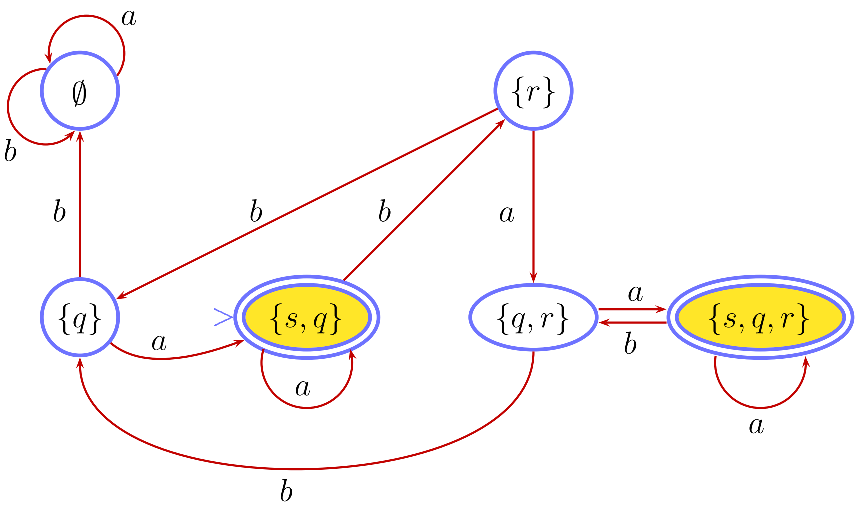


Subset construction example: the result

NFA:

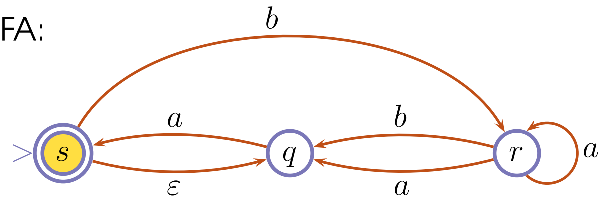


DFA:



Subset construction example: the result's transition table

NFA:



	a	b	ε
s	–	r	q
q	s	–	–
r	$q,\ r$	q	–

DFA: initial state: $\{s, q\}$ favourable states: $\{s, q\}$, $\{s, q, r\}$

	a	b
$\{s, q\}$	$\{s, q\}$	$\{r\}$
$\{r\}$	$\{q, r\}$	$\{q\}$
$\{q, r\}$	$\{s, q, r\}$	$\{q\}$
$\{q\}$	$\{s, q\}$	$\emptyset$
$\{s, q, r\}$	$\{s, q, r\}$	$\{q, r\}$
$\emptyset$	$\emptyset$	$\emptyset$

State minimisation problem

An important field where finite automata are being used is *hardware design*: some components of hardware (chips) are based on simulating DFAs.

- Given a task, an NFA is designed (since it is easier to design than a DFA).
- Then this NFA is turned to a DFA doing the same thing (using the subset construction).

  However, the number of states of the resulting DFA may increase significantly:

  $$\boxed{2^{\textbf{number of states of the original NFA}} \quad \text{at the worst case}}$$

- Next, the resulting DFA is 'minimised', to obtain a DFA that has the *minimum* possible number of states for the given task.

  (There are various methods for doing this, see recommended literature).
- Finally, the obtained minimal DFA is hard-wired.

Regular expressions: what for?

- They are used in any task that require *pattern matching*:
  - search engines,
  - protein analysis in bioinformatics,
  - search and replace functionalities in word processors and text editors,
  - text processing utilities like `sed` and `awk` in Unix and Linux.
- Many programming languages provide various regular expression capabilities.
- Lexical analysers in compilers also use regular expressions.
- . . .

Finite representation of languages

Consider the following language:

$L =$ all strings of $\Box$s and $\Diamond$s that have two or three occurrences of $\Diamond$, the first and second of which are not consecutive

Can we describe this **infinite language** with some kind of **finite 'pattern'?**

- Strings in L can start with any number (possibly none) of $\Box$s: $\underline{\Box^*}$
- Then comes the first $\Diamond$.

  It should be alone, so must be followed by at least one $\Box$: $\underline{\Box^*\Diamond\Box}$
- Then any number (possibly none) of $\Box$s: $\underline{\Box^*\Diamond\Box\Box^*}$
- Then comes the second $\Diamond$: $\underline{\Box^*\Diamond\Box\Box^*\Diamond}$
- Then
  - **either** no more $\Diamond$s but there can be any number (possibly none) $\Box$s ,
  - **or**, after some (possibly none) $\Box$s , there is a third $\Diamond$ somewhere, followed by any number (possibly none) $\Box$s :

$\underline{\Box^*\Diamond\Box\Box^*\Diamond(\Box^* \cup \Box^*\Diamond\Box^*)}$ $\rightsquigarrow$ $\boxed{\underline{\textbf{regular expression}}\text{ representing } L}$

From language to regular expression: Example 2

L = all the strings consisting of some number (possibly none) of a s, followed by some number (possibly none) of b s

$= \{\varepsilon, a, b, aa, bb, ab, aaa, aab, bbb, abb, \dots, aaaabbbbbb, \dots\}$

Let's try: $\underline{a^*b^*}$

$\rightsquigarrow$ L is represented by the regular expression a^*b^*

Notation: $\boxed{L \;=\; \text{Language\_of}(a^*b^*)}$

read: " the language of a^*b^* is L "

or " the language represented by a^*b^* is L "

From regular expression to language

Example 3: Language_of$\big(a(a^* \cup b^*)\big) = ?$

$=$ all words starting with a, followed by either a (possibly empty) word of as, or a (possibly empty) word of bs

$= \{a,\ aa,\ ab,\ aaa,\ abb,\ aaaa,\ abbb,\ aaaaaa,\ abbbbbbb,\ \dots\}$

Example 4: Language_of$\big(a(a \cup b)^*\big) = ?$

$=$ all words starting with a, followed by any word over $\{a, b\}$

$=$ all words over the alphabet $\{a, b\}$ starting with a

$= \{a,\ aa,\ ab,\ aaa,\ aab,\ aba,\ abb,\ aaaaaa,\ abbaabab,\ aabbababba,\ \dots\}$

The two examples are NOT the same!

From regular expression to language: Example 5

$$\boxed{\text{Language\_of}\big((b \cup aaa^*)^*\big) \; = \; ?}$$

- Let's start with $\underline{\text{Language\_of}(aaa^*)}$: all words of as of length ≥ 2

  $= \{aa, \; aaa, \; aaaa, \; aaaaa, \; \ldots\}$

- $\underline{\text{Language\_of}(b \cup aaa^*)}$: as above, plus the word b

  $= \{b, \; aa, \; aaa, \; aaaa, \; aaaaa, \; \ldots\}$

- $\underline{\text{Language\_of}\big((b \cup aaa^*)^*\big)}$: we can take any number (possibly none) of words from $\text{Language\_of}(b \cup aaa^*)$ and concatenate them

Some questions checking our understanding of this language:

- Is there a word of as and bs that is not in $\text{Language\_of}\big((b \cup aaa^*)^*\big)$?
- Do the words ε, $aabbaaabab$, and $bbbabbaaabaa$ belong to the language $\text{Language\_of}\big((b \cup aaa^*)^*\big)$?

Summary: describing regular expressions by recursion

Each regular expression (over an alphabet S) is a string consisting of

- symbols from S,
- plus symbols from $\cup$, $*$, $($, $)$, ε, $\emptyset$.

The (infinite) set of all **regular expressions over S** is described recursively:

- *Basis:* $\emptyset$, ε and each symbol in the alphabet S are regular expressions.
- *Recursive steps:* If R and Z are regular expressions, then so are $(R \cup Z)$, (RZ), and (R^*).

FOR EXAMPLE:

- Over $S = \{a, b\}$: $\emptyset$, ε, $((a^*)(b^*))$, $(((a((a \cup b)^*))b)(b^*))$
- Over $S = \{x, y\}$: $\emptyset$, ε, $(((x(y^*))y)(y \cup z))$, $(((x(y^*))y)(y \cup z))$
- Over $S = \{\Diamond, \Box\}$: $\emptyset$, ε, $((\Box \cup (\Diamond^*))((\Diamond\Box)^*))$

Regular expressions: notational conventions

These brackets are pretty incomprehensible. So we introduce some conventions:

- We omit the outermost brackets.
- We omit the brackets when concatenate expressions,

  so e.g. we write $ababb$ instead of $(((ab)(ab))b)$.
- * 'binds tighter' than concatenation,

  so e.g. we write aab^* instead of $(aa)(b^*)$.

For example: (cf. the previous lecture slide)

$$a^*b^*, \qquad a(a \cup b)^*bb^*, \qquad xy^*y(y \cup z), \qquad (\Box \cup \Diamond^*)(\Diamond\Box)^*$$

But take care: say, $(aa)^*b$ and aa^*b are NOT the same!

Always use brackets if in any doubt!

From regular expression to language: Example 6

$\boxed{\text{Language\_of}\big((b \cup a)aa^*\big) \ = \ ?}$

Let's do it, 'from inside out':

- $\underline{\text{Language\_of}(b \cup a)} \ = \ \{a, b\}$
- $\underline{\text{Language\_of}\big((b \cup a)a\big)} \ = \ \{aa, ba\}$
- $\underline{\text{Language\_of}\big((b \cup a)aa^*\big)} \ =$ all words starting with aa followed by a (possibly empty) word of a s, and all words starting with ba followed by a (possibly empty) word of a s .

Compare this language with $\text{Language\_of}(b \cup aaa^*)$ (see Example 5).

Brackets DO MATTER: the two languages are NOT the same!

From regular expression to language: Example 7

$$\boxed{\text{Language\_of}\big(\varepsilon \cup c^*(a \cup bc^*)\big) \;=\; ?}$$

all the strings that are either empty or start with some (possibly none) cs, followed by either an a, or a b followed by some (possibly none) cs

$= \{\varepsilon,\ a,\ b,\ bc,\ bcc,\ ca,\ cb,\ cbcc,\ \dots\}$

$\underline{\text{NOT in Language\_of}\big(\varepsilon \cup c^*(a \cup bc^*)\big):}$ $\quad ab,\ \ aa,\ \ caac,\ \ \dots$

Summary: regular expressions represent languages

Each regular expression over some alphabet S **represents** a language over S.

These are two different things:

a regular expression R ⟷ the language R represents: Language_of(R)

a string consisting of symbols from S and $\cup,\ *,\ (,\ ),\ \varepsilon,\ \emptyset$

a set of words over S

FOR EXAMPLE:

$1(1 \cup 2)^*$

Language_of$\big(1(1 \cup 2)^*\big)$ = all words of 1 s and 2 s starting with 1

$(a \cup b)bba$

Language_of$\big((a \cup b)bba\big) = \{abba,\ bbba\}$

finite 'pattern'

words following the pattern (can be infinitely many)

Summary: how regular expressions represent languages

Recursive description of the language $\boxed{\text{Language\_of}(R)}$ represented by the regular expression R:

We follow the recursive description of regular expressions on lecture slide 272.

Basis: $\text{Language\_of}(\emptyset) = \emptyset$

$\text{Language\_of}(\varepsilon) = \{\varepsilon\}$

$\text{Language\_of}(a) = \{a\}$ for any symbol a in the alphabet S.

Recursive steps: If R and Z are regular expressions, then

- $\text{Language\_of}(R \cup Z) =$ all words that belong either to $\text{Language\_of}(R)$ or to $\text{Language\_of}(Z)$
- $\text{Language\_of}(RZ) =$ any word from $\text{Language\_of}(R)$ followed by any word from $\text{Language\_of}(Z)$
- $\text{Language\_of}(R^*) =$ any word from $\text{Language\_of}(R)$
  followed by any word from $\text{Language\_of}(R)$
  followed by any word from $\text{Language\_of}(R)$ $\ldots$

  the number of iterations is arbitrary (possibly none)

Regular languages

A **regular language**

is *any* language that can be represented by a regular expression.

In other words, a language L is **regular** if $L = \text{Language\_of}(R)$ for some regular expression R.

For instance, all the languages in Examples 1–7 above are regular.

$\boxed{\text{NOT every language is regular!}}$ (we will see later)

- regular expression = pattern
- regular language = set of words following a pattern

Regular languages and non-deterministic finite automata

(1) **There is a general 'mechanical' procedure** $\mathcal{P}_1$ **that converts any regular expression** R **to an NFA** A_R **such that** $L(A_R) = \text{Language\_of}(R)$ **.**

So every regular language is accepted by some automaton.

(We will discuss this algorithm $\mathcal{P}_1$ on the next lecture slides.)

Moreover, there is a general way back:

(2) **There is a general 'mechanical' procedure** $\mathcal{P}_2$ **that converts any NFA** A **to a regular expression** R_A **such that** $\text{Language\_of}(R_A) = L(A)$ **.**

So every language that is accepted by some automaton is regular.

(Algorithm $\mathcal{P}_2$ is out of our scope, see the literature if interested.)

Important:

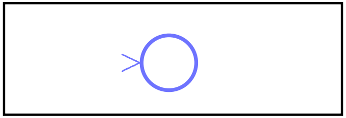
Regular languages are precisely those languages that are accepted by finite automata.

From regular expressions to NFA

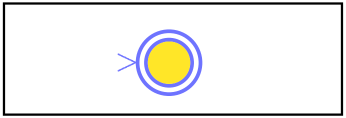
The procedure $\mathcal{P}_1$ which
converts any regular expression R to an NFA A_R such that $L(A_R) = \mathsf{Language\_of}(R)$
operates along the recursive description of the regular expression R
(again, see lecture slide 272):

Basis cases:

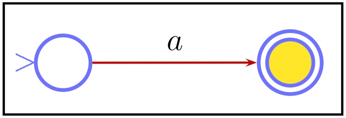
- If $R = \emptyset$. Then $\mathsf{Language\_of}(R) = \emptyset$. A_R :



- If $R = \varepsilon$. Then $\mathsf{Language\_of}(R) = \{\varepsilon\}$. A_R :

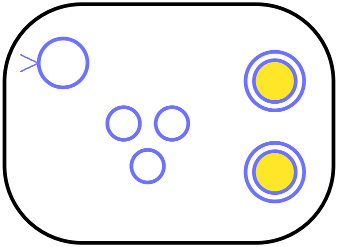


- If $R = a$. Then $\mathsf{Language\_of}(R) = \{a\}$. A_R :

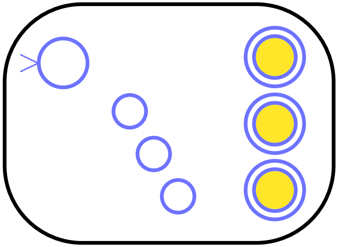


Recursive cases: Automaton accepting Language_of$(R \cup Z)$

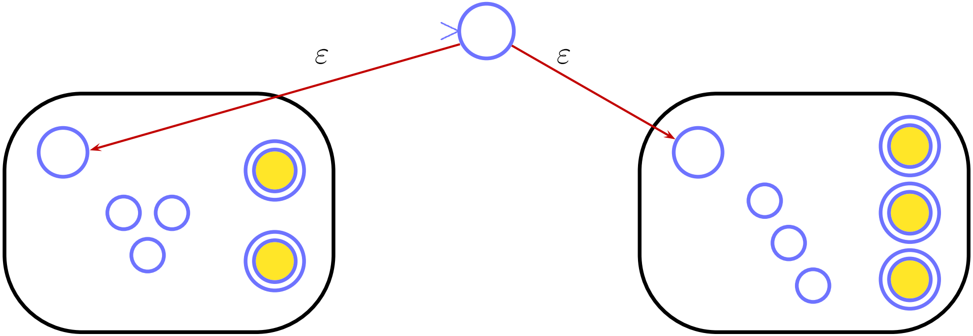
$\boxed{A_R:}$ accepts Language_of(R)



$\boxed{A_Z:}$ accepts Language_of(Z)

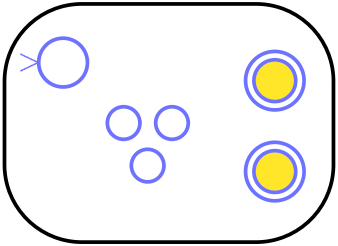


$\boxed{A_{R\cup Z}:}$ accepts Language_of$(R \cup Z)$

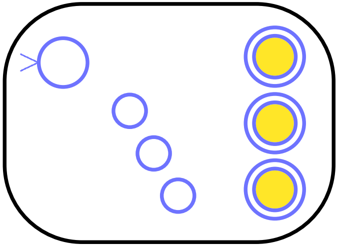


Recursive cases: Automaton accepting Language_of(RZ)

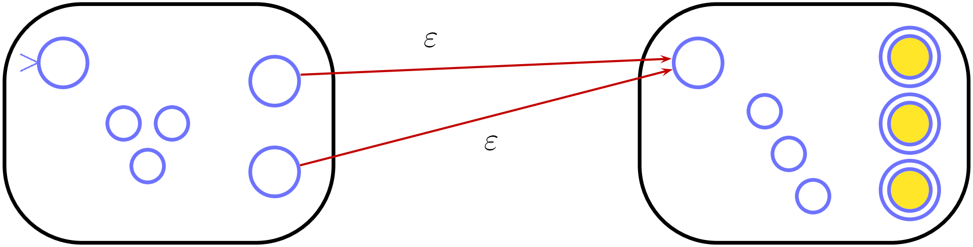
$\boxed{A_R:}$ accepts Language_of(R)



$\boxed{A_Z:}$ accepts Language_of(Z)

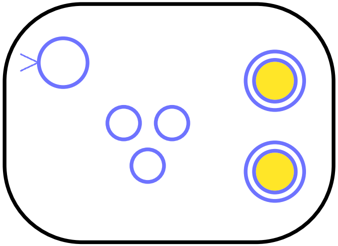


$\boxed{A_{RZ}:}$ accepts Language_of(RZ)

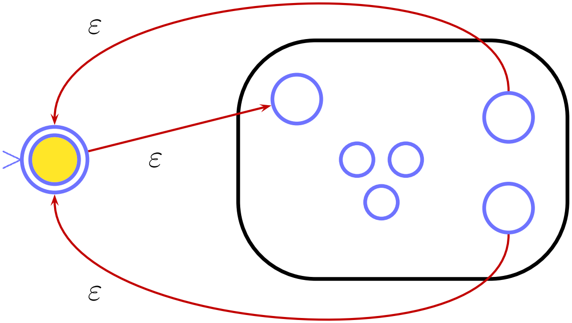


Recursive cases: Automaton accepting Language_of(R^*)

$\boxed{A_R:}$ accepts Language_of(R)



$\boxed{A_{R^*}:}$ accepts Language_of(R^*)



Example: how the procedure works

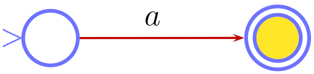
We apply the procedure to the regular expression

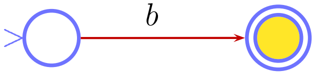
$$\big((a \cup ab)^* ba\big)^*$$

We construct step by step (going ‘inside out’) an NFA A such that

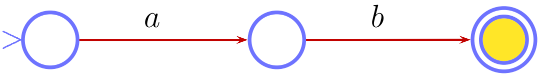
$$L(A) \;=\; \text{Language\_of}\big(((a \cup ab)^* ba)^*\big)$$

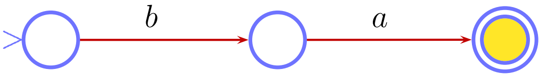
Step 0: automata accepting Language_of(a) and Language_of(b)



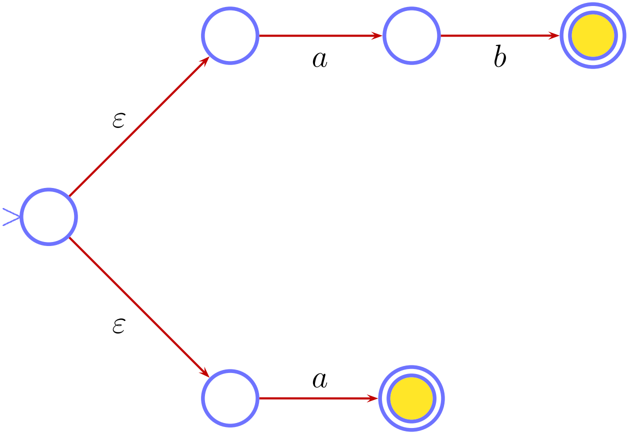


Step 1: automata accepting Language_of(ab) and Language_of(ba)

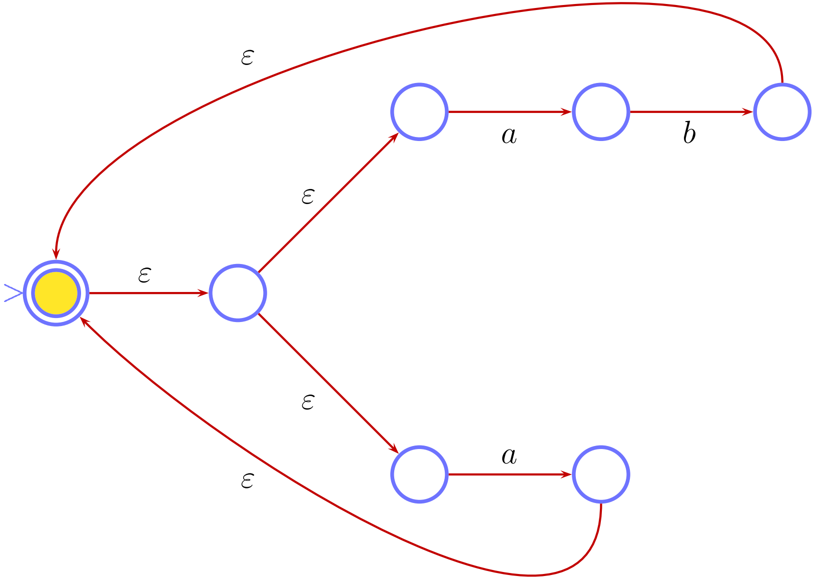




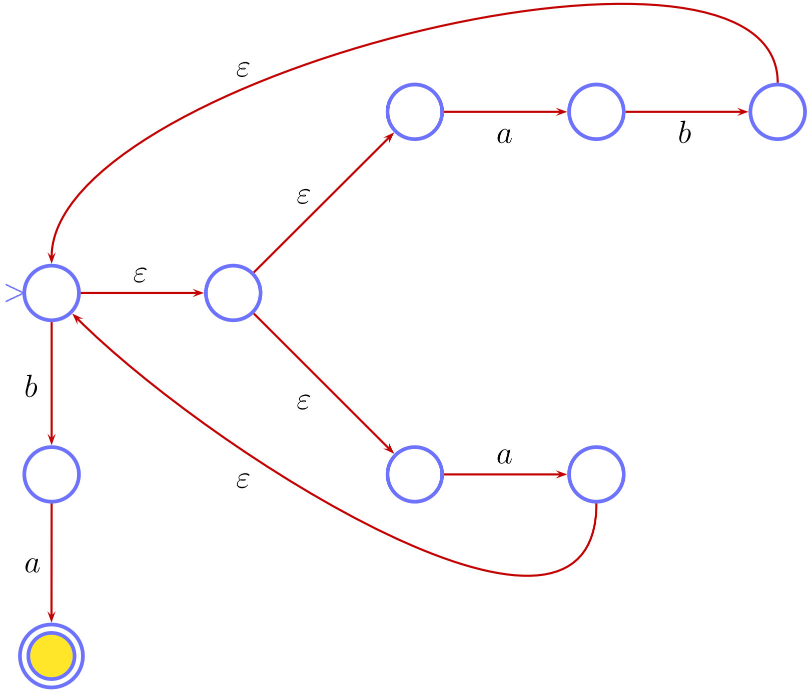
Step 2: automaton accepting Language_of$(a \cup ab)$



Step 3: automaton accepting Language_of$\big((a \cup ab)^*\big)$

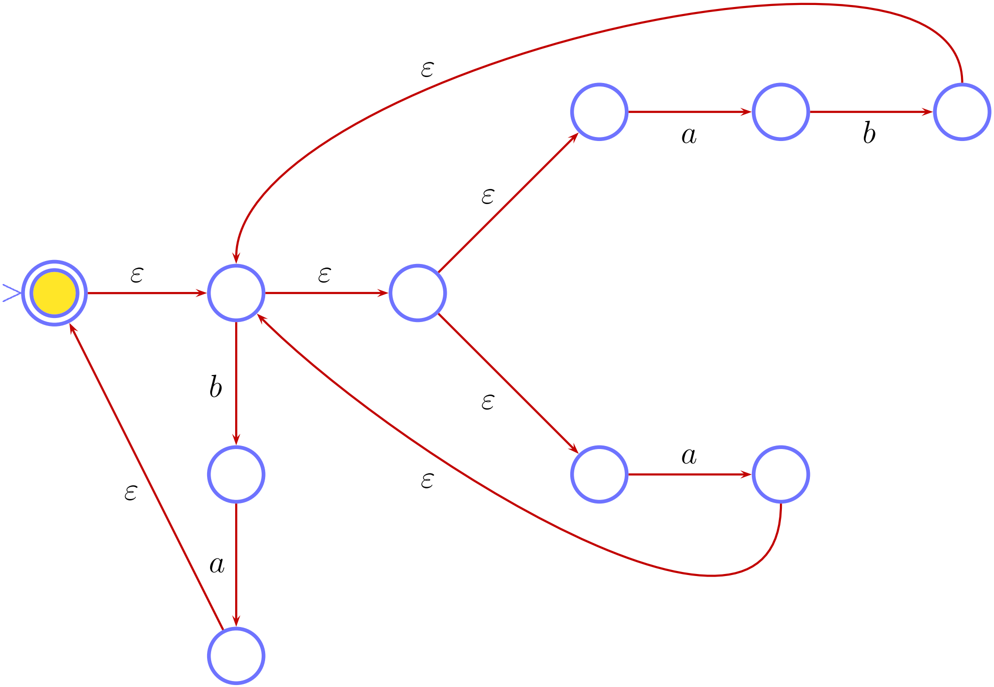


Step 4: automaton accepting Language_of $\big((a \cup ab)^* ba\big)$



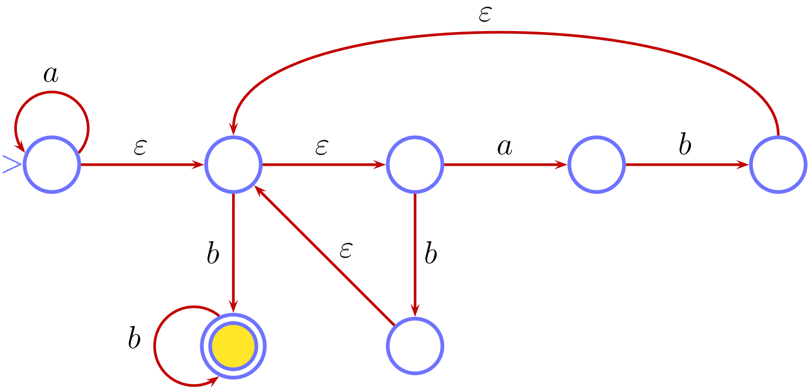
Final step of the procedure

Automaton accepting Language_of$\left(\left((a \cup ab)^*ba\right)^*\right)$:



Another example

Automaton accepting Language_of$\big(a^*(b \cup ab)^* bb^*\big)$:



Finite automata and regular languages: summary

- Finite automata may be regarded as programs that use **fixed amounts of memory** (represented by states) regardless of the input.
- Finite automata can be used as **recognition devices:** they accept certain inputs and reject others.
- **Nondeterminism** does not increase the computational power of finite automata, but nondeterministic automata are easier to design than **deterministic** ones.
- The languages accepted by finite automata are precisely the **regular** ones.

Nonregular languages: an example

Consider the following language over the alphabet $\{a, b\}$:

$$\begin{aligned} L &= \text{all words starting with a word of } a\text{'s} \\ &\quad\ \text{followed by an equal-length word of } b\text{'s} \\ &= \{a^n b^n \mid n = 0, 1, 2, \ldots\} \end{aligned}$$

Suppose that a finite automaton A tries to recognise words in this language. Then A must store the <u>entire sequence of a's</u> before the first b shows up. Otherwise A will not be able to compare the length of the coming word of bs with the length of the prefix of as.

As each automaton is capable for a fixed finite amount of storage with the help of its states, no automaton A exists such that $L(A) = L$.

There is a precise mathematical proof justifying this *informal* argument:

$\boxed{\text{this language } L \text{ is indeed } \textbf{not regular}}$