4CCS1FC1 Foundations of Computing

1ST YEAR SEMESTER 1

Franch Tapia, Alex
KING'S COLLEGE LONDON | SEP - DEC

4CCS1FC1 Foundations of Computing

Lecturer

Agi Kurucz

- agi.kurucz@kcl.ac.uk
- www.inf.kcl.ac.uk/staff/kuag
- Room S1.17 Office hours: Tuesday 1-2:30; Wed 12:30-2

Assessment

- Midterm Class Test (10%) [Only first five weeks will be covered]
- Exam (90%) in January
- 4 Optional pieces of coursework are available – can ask Agi to mark.

Sets

Basics

Set: is a collection of things called **elements**. TIP: It's a good idea to think of it as a plastic bag, it has stuff (elements) inside.

Simple notation

$x \in S$ – x **is** an element in S

$x \notin S$ – x **is not** an element in S

$x, y \in S$ – x and y are **both** elements in S

Describing sets

By listing

In order to describe a set by listing, we can **explicitly name its elements**. (JUST NAME THEM BRA, NO FURTHER NEED TO EXPLAIN)

But remember that **anything** can be an elements of a set. Even a set can be an element of another set.

$B = \{x, \{x, y\}\}$ – These are completely different elements.

The most important thing about a set is

WHAT IS IN IT AND WHAT IS NOT IN IT.

Things like the number of times an element appears or the order doesn't matter. They all describe the **same set**.

The Empty Set

A set with no elements is called an **empty set** $\emptyset$, it is also sometimes defined { }. It can also be an element of a set. A good way to think of it, is that it's just like a bag, it may not contain anything but it's there.

PRO TIP Bear in mind that this $\{\emptyset\}$ is not an empty set, rather a set with an empty set in it.

Singleton

A **singleton** is a set which only has one element in its set.

Equality of Sets

Sets are **equal** when they have the same elements.

$$A = B$$

Or not

$$A \neq B$$

Egs

$\{h, f, j\} = \{j, h, f\}$

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

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

By properties

When elements within sets have certain properties, we can simplify the way we express them.

$$S = \{x \mid x\ has\ property\ P\}$$

Read as: "**S** is the set of all ***x*** which have property **P**."

In the case the **S** came from a larger set **A** we can say:

$$S = \{x \in A \mid x\ has\ property\ P\}$$

Eg

Describing the Odd set within the Integers (there are different ways)

$Odd = \{x \in Z \mid x \text{ is } Odd\}$

$Odd = \{x \mid x = 2k + 1 \text{ for some integer } k\}$

$Odd = \{2k + 1 \mid k \in Z\}$

By induction

It consists of three steps.

1. **Basis:** Specify one or more elements of a set, for example S.
2. **Inductive Step:** Give one or more rules to construct new elements in S from existing elements in S.
3. **Closure:** State that S consists only of elements obtained by the basis and inductive steps and nothing else in S.

E.g

The Natural Numbers

Basis: $0 \in N$

Inductive Step: $If\ n \in N\ then\ N + 1 \in N$

The set $A = \{3k + 1 \mid k \in N\}$

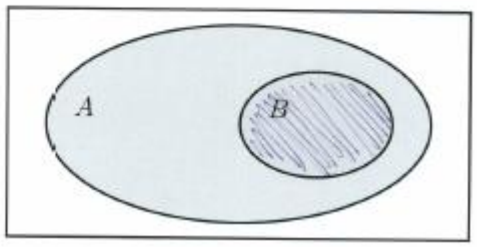
Basis: $1 \in A$

Inductive step: $If\ x \in A\ then\ x + 3 \in A$

Subsets

A set B is a subset of A, if every element in B is also in A.

$$B \subseteq A$$



Egs

$$\{a, c\} \subseteq \{a, b, c, d\} \qquad \{0,1,5\} \subseteq N$$

We always know that for every set S:

$$S \subseteq S \qquad \emptyset \subseteq S$$

If B is a subset of A, and there is some element in A that is not ib B, then we call it a **proper subset** of A, and write $B \subset A$.

$$B \subset A, if\ B \subseteq A\ but\ B \neq A$$

How to show that $A \subseteq B$

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

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

Show that $A \subseteq B$

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

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

Which is either 43 or 47.

We can have:

$43 = 4 \cdot 10 + 3,$ the choice of k = 10, shows that $43 \in B$.

$47 = 4 \cdot 11 + 3$, the choice of k = 11 shows that $47 \in B$ as well.

Show that $A \subset B$

We have to find an element which is $x \in B$ such that $x \notin A$.

For example:

$0 \in N\ and\ 3 = 4 \cdot 0 + 3, we\ have\ that\ 3 \in B\ but\ we\ know\ that\ 42 \nleq 3, so\ 3 \notin A.$

How to show that $A \nsubseteq B$

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

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

Show that $A \nsubseteq B$

We have to find an element in $x \in A$ such that $x \notin B$

If we take $x = 4$

$4 = 3 \cdot 1 + 1.\ So\ k = 1\ shows\ that\ 4 \in A$

We now need to show that there is no $k \in N$ for which B is true.

$k = 0,\ for\ B\ this\ means\ taht\ B = 4 \cdot 0 + 1 = 1 \neq 4$

How to show that $A = B\ or\ A \neq B$

$$A = B\ means\ that\ A \subseteq B\ and\ B \subseteq A$$

If the task is to show that $A = B$, then we need to show that **both** $A \subseteq B$ ***and*** $B \subseteq A$

If the task is to show that $A \neq B$, then we either need to find an element in A that is **not** in B, or find an element in B that is not in A.

Power Sets

A **power set** is the set of all subsets of a set S.

$$P(S)$$

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

RULES: $\emptyset \in P(S)$ and $S \in P(S)$

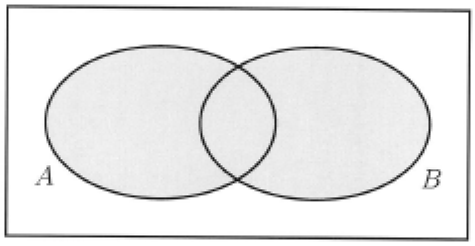
Egs: $P(\{Joe, Tuesday\}) = \{\emptyset, \{Joe\}, \{Tuesday\}, \{Joe, Tuesday\}\}$

Set Operations

Union

$$A \cup B = \{x \mid x \in A \; or \; x \in B\}$$

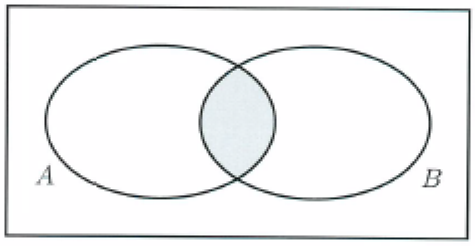
It consists of those elements which are either in A, B or **both**.



Intersection

$$A \cap B = \{x \mid x \in A \; and \; x \in B\}$$

Consists of those elements that are both in A and in B.

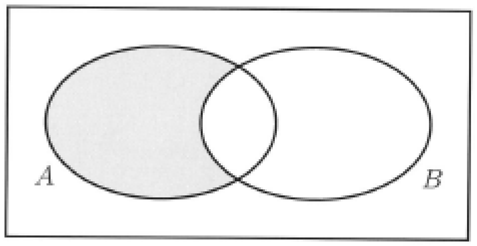


If $A \cap B = \emptyset$, then we say that A and B are **disjoint**.

Difference

$$A - B = \{x \mid x \in A \; and \; x \notin B\}$$

Also called the **complement of B with respect to A.**

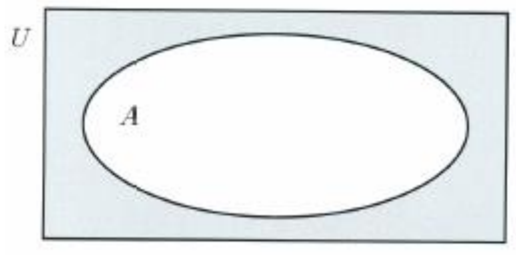


Absolute complement

In some operations we might consider working with a subset of a **universal set** *U*.

Given a universal set U and $A \subseteq U$, the complement of A is the set:

$$\bar{A} = U - A = \{x \in U \mid x \notin A\}$$



Sequences

Basics

Sequence: a list of things, taken in a certain order.

In a sequence the order of listing **does** matter and the order of occurrences **does** matter as well.

Tuple: a finite sequence. Two tuples are the same if they have the **same length** and **the corresponding elements are the same**

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

Of sets A and B

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

It consists of ordered pairs.

Of sets $A_1, A_2, \ldots, A_k$

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

This Cartesian product consists of k-tuples.

Binary Relations

A binary relation from A to B is *any subset of the Cartesian Product A x B.*

We use notation aRb to denote that $(a,b) \in R$ and say that a is **related** to b.

The contradiction is the same in which R has a line through it.

Relations on A Set

A set can have a relation on itself. For instance:

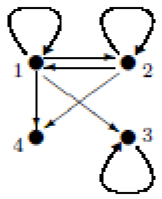
For a set of **Z** integers:

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

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

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

Representing relations



directed graph

Or, by a table:

R	1	2	3	4
1	X	X	X	X
2	X	X		X
3			X	
4				

Or, by its **matrix:**

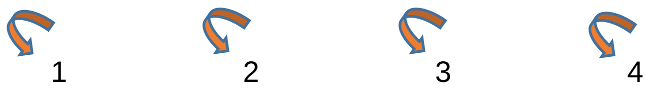
$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Note: **for the table** this is how the relation is represented (row, column).

Properties of relations

Reflexive: means that for all $a \in S$, aRa or $(a,a) \in S$

In the arrow diagram it is represented by all elements in the set having a loop or circle on itself.



Some examples are $\leq, \geq,$ `$divisibility$´

Not reflexive is when only some have loops.

Irreflexive: if $For\ a\ set\ A,\ (a,a) \in R\ for\ every\ element\ a \in A$.

No numbers have loops.

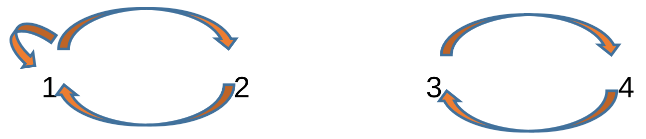
1 2 3 4

Some examples are $<, >, \neq$ on **N** or on **Z**.

Symmetric: if aRb, then bRa

In the arrow diagram this is represented by double arrows, back and forth. Not necessarily for **all** but for all that are present in our sequence.

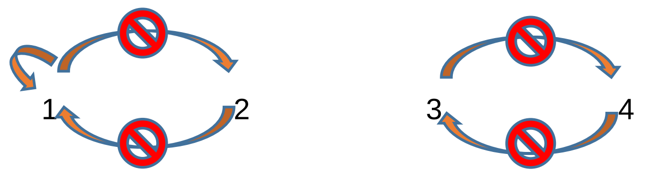
E.g $R = \{(1,1), (1,2), (2,1), (3,4), (4,3)\ on\ \{1,2,3,4\}$



Antisymmetric: has all single arrows.

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

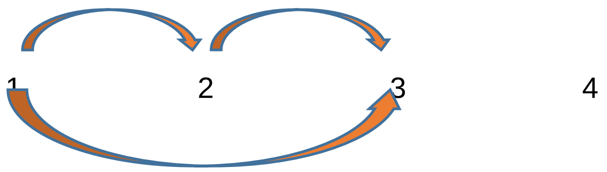
Basically there cannot be double arrows but single reflexive arrows are allowed.

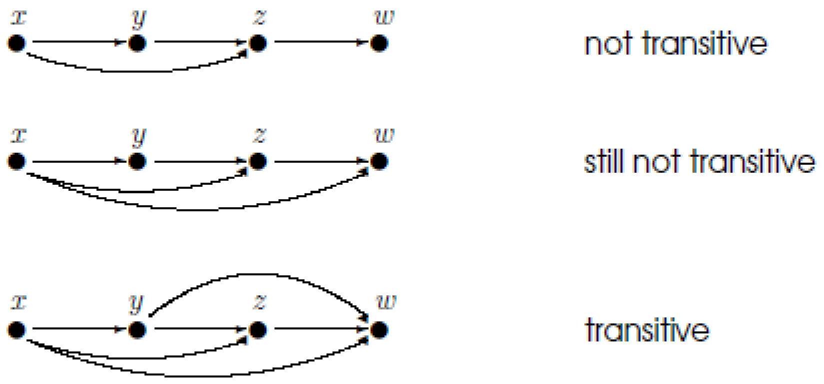


For example: $<, >, \leq, \geq\ on\ \boldsymbol{N}, \boldsymbol{Z}, \boldsymbol{Q}\ or\ \boldsymbol{R}$

Transitive: all steps that can be done in two steps have to be able to be done in one step in some way.

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





Some examples: $<, >, \leq, \geq$ on $\mathbf{N}$, $\mathbf{Z}$, $\mathbf{Q}$, or $\mathbf{R}$

Transitive closure

The transitive closure of R which is R^* will be the transitive set of tuples in which all two steps can be done in one step.

To find out so, we can use **Warshall's algorithm**

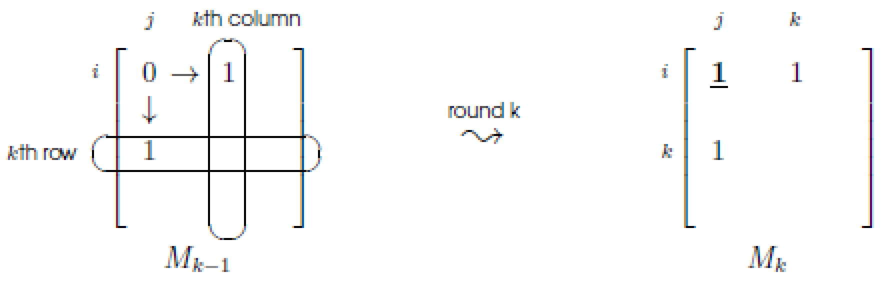
- Given a relation R on a set with n elements, we begin with its $n \times n$ matrix $\boxed{M_0}$.
- There are n rounds.

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

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

  $\boxed{M_n}$ 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** 0s to 1s:



Equivalence Relations

- reflexive,
- symmetric, and
- transitive.

FOR EXAMPLE:

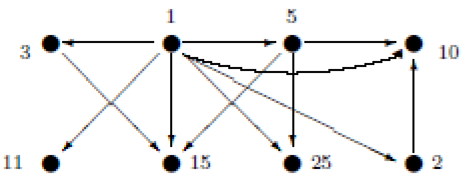
- $=$ on any set,
- $\equiv_4 = \{(x,y) \in \mathbf{Z} \times \mathbf{Z} \mid x = y \pmod 4\}$ on $\mathbf{Z}$

Partial Orders

- reflexive,
- antisymmetric, and
- transitive.

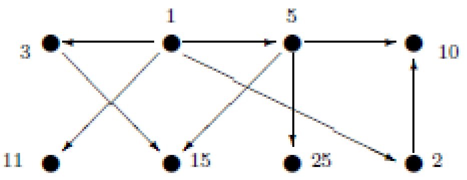
FOR EXAMPLE: $\leq$, $\geq$, and 'divisibility' on $\mathbf{N}^+$

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 $\begin{tikzcd} \overset{1}{\bullet} \arrow[r] & \overset{5}{\bullet} \arrow[r] & \overset{10}{\bullet} \end{tikzcd}$ is part of our diagram, then we know that we must also have $\begin{tikzcd} \overset{1}{\bullet} \arrow[r] \arrow[rr, bend right] & \overset{5}{\bullet} \arrow[r] & \overset{10}{\bullet} \end{tikzcd}$

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

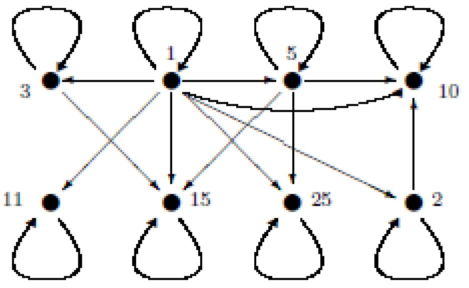


If we know that a relation is a partial order, then there is a more 'economical' way of representing it than by a directed graph.

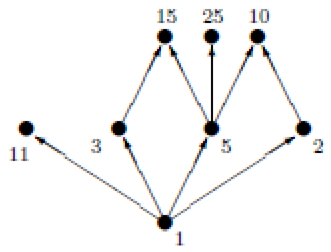
Say, take the 'divisibility' relation on the set $\{1, 2, 3, 5, 10, 11, 15, 25\}$:

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

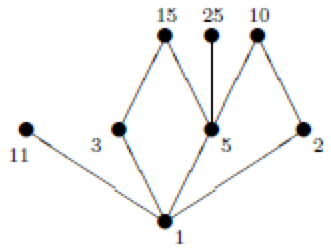
As this is a relation, it can be represented by a directed graph:



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!)

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 is `comparable' this way or the other according to R).

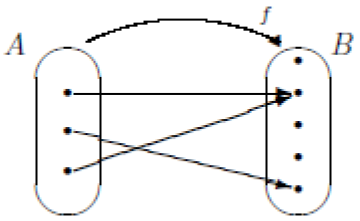
FOR EXAMPLE:

- $\leq$ and $\geq$ on **N**, **Z**, **Q**, or **Z**
- BUT: `divisibility' and $\subseteq$ are partial orders, but NOT linear orders

Functions

Basics

A **function** from A to B is a rule that associates with each element of A exactly one elements of B.



If $x \in A$ and $y \in B$, then $f(x) = y$; we can say "*f of x is y*" or *"f maps x to y"* or *"value of f at x is y".*

Can also be written as $f: A \to B$, we can A the **domain** of f, and B the **codomain** of f.

Rules for something to classify as a function:

- **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

Describing functions

- By listing the associations e.g $f(a) = 1$.
- By drawing points and arrows.
- By drawing a graph.

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

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

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

Functions with multiple arguments

If the domain of a function f is a Cartesian product we say f has **arity** n (depends on the Cartesian product multiplier) or that it has n arguments.

A function f with two arguments is called a **binary function**, they can be written in two ways:

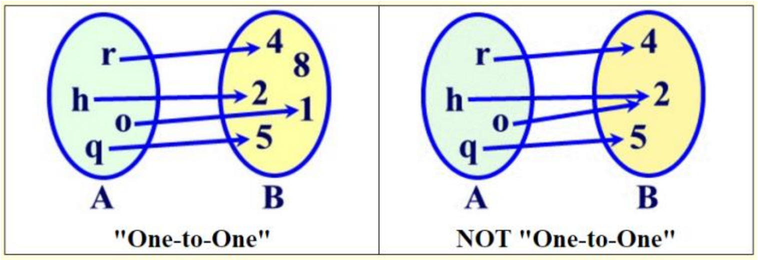
$$f(x, y) = z \text{ or } x\ f\ y = z$$

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

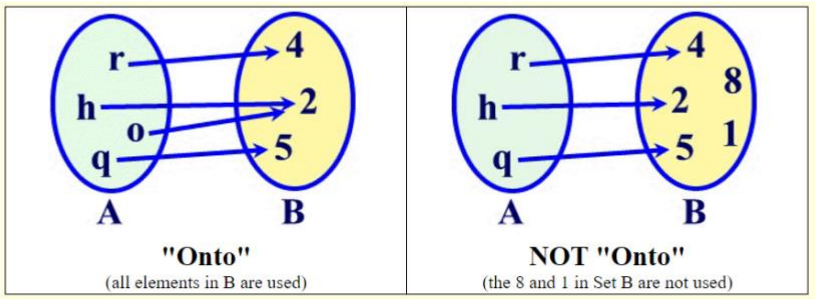
$$\{(a, b) \in A \times B \mid f(a) = b\}$$

Properties of functions

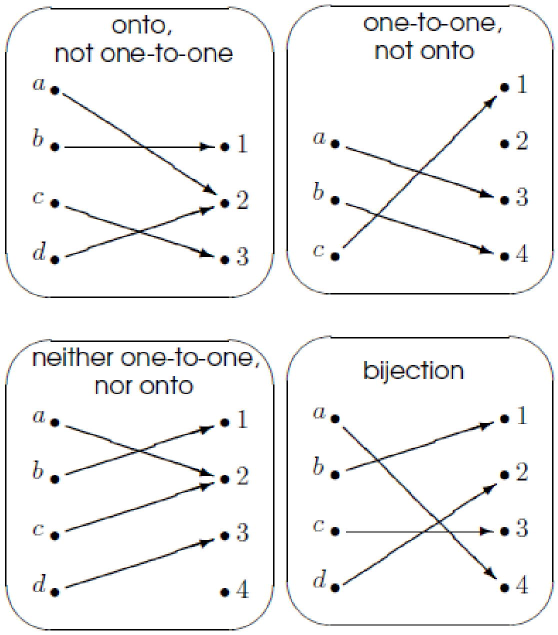
- A function $f : A \to B$ is one-to-one if it maps a **distinct** element of A to a **distinct** element of B.



- A function $f : A \to B$ is onto "*si todas las salidas tienen entrada*".


r
h
o
q
4
2
5
A
B
"Onto"
(all elements in B are used)
r
h
q
4
2
5
8
1
A
B
NOT "Onto"
(the 8 and 1 in Set B are not used)

If a function is both onto and one-to-one it's called a **bijection**.



If a function is a bijection the **inverse**

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

Combining functions

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

$$\begin{tikzcd}
A \arrow[r, bend left, "g"] \arrow[rr, bend right, "f \circ g"'] & B \arrow[r, bend left, "f"] & C
\end{tikzcd}$$

Countable sets

A set is countable if its **finite** or if there's a bijection between it and the natural numbers.

Note that the Cartesian product of natural numbers is also countable.

Non Countable sets

For instance, the $P(N)$ is not countable.

Size of sets

The cardinality, or size of a set is desfined by $|S|$.

For instance, if $S = \{1,2,3,4\}$ then $|S| = 4$. The cardinality of the empty set is zero.

$$|\emptyset| = 0$$

The sum rule

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

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

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

The inclusion exclusion principle

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

We count the ones that appear in both twice, hence we have to take these away.

For three sets:

7Th

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

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

The product rule

To calculate the cardinality of the Cartesian product of many sets we do the following:

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

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:

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

Counting the subsets of a finite set

We can create a set S and then a subset of this A. When creating this we can choose each element to either be **in** or **out**.

Therefore, that means that:

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

The pigeonhole principle

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

Four ways of selecting items

- Order matters, repetitions not allowed (combination)
- Order matters, repetitions allowed (combination)
- Order doesn't matter, repetitions not allowed (permutation)
- Order doesn't matter, repetitions allowed (permutation)

Order matters, repetitions not allowed

How many ways can we select $\boldsymbol{k}$ persons from a group of $\mathbf{n}$ people in different ways?

There are n ways to select the first person, $n-1$ the second person up until $n-(k+1)$ ways to select the k^{th} person.

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

SPECIAL CASE: if $k = n$. If you want to know how many ways you can order all the people.

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

Order matters, repetitions allowed

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

There are n ways of choosing the first letter, and n ways to choose the second letter, etc.

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

Order does not matter, repetition not allowed

As seen if order of selection does matter, there are $\frac{n!}{(n-k)!}$ Ways to select people, but since the **order doesn't matter**, that means that (1,2) and (2,1) are the same thing. Hence we have to take away some results.

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

$\underline{n \text{ choose } k}$

A tip to solve this is we can do:

$$\frac{k\ times\ backward\ steps}{k\ times\ forward\ steps}$$

In other words:

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

There are called binomial coefficients and its interesting to see that due to the pascal triangle we know that:

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

So:

$$\binom{6}{2} \equiv \binom{6}{4}$$

Order does not matter, repetitions 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: $\boxed{A\,A\,A\,A \mid\mid}$

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

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

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

Probability

Basics

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.

Complementary event

$$p(\bar{E}) = 1 - p(E)$$

Union of events

$$p(A \cup B) = p(A) + p(B) - p(A \cap B)$$

Conditional probability

$$p(E|F) = \frac{p(E \cap F)}{p(F)}$$

Independence

Two events are independent if the outcome of one does not affect the outcome of the other.

In that case $p(E|F) = p(E)\ and\ p(F|E) = p(F)$

Using the formula above.

$$p(E|F) = \frac{p(E \cap F)}{p(F)} = p(E)$$

Hence if we rearrange we see that:

$$p(E \cap F) = p(E) \cdot p(F)$$

Bayes' Theorem

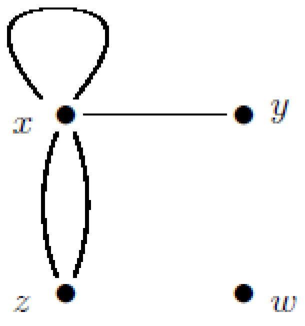
$$\boxed{p(F \mid E) = \frac{p(E \mid F)p(F)}{p(E \mid F)p(F) + p(E \mid \bar{F})p(\bar{F})}}$$

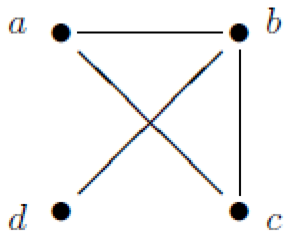
Easy! At the stop we swap and multiply by the condition, bottom the same plus the condition denied.

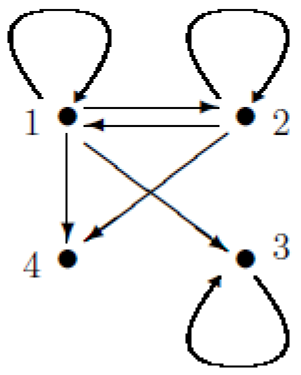
Graphs

Graph: drawings with dots and lines or arrows.

The dots are called **vertices** (or nodes) and the arrows are called edges (in order).







Directed Graphs

Simple Graph

Undirected edges, no multiple edges, no loop edges.

Multigraph

Undirected edges, multiple edges, loops on edges.

Features

If two points are connected, they are **adjacent**.

The edge that connects them is **incident with** the two points.

Degree of the vertex: the number of edges that come out of it.

A vertex of degree zero is called **isolated**.

A vertex of degree one is called **pendant**.

Handshake theorem:

$$\mathit{number\ of\ edges} = \frac{\mathit{sum\ of\ the\ degrees\ of\ the\ vertices}}{2}$$

Directed graph

Directed edges, no multiple edges, loops on edges.

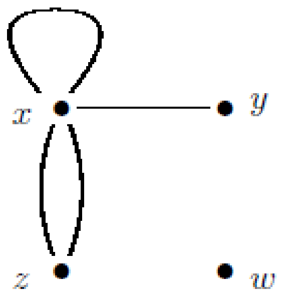
If an edge connects two points, they are **adjacent**.

The one on the left of the arrow is the **start/initial vertex** and a **terminal/end vertex**.

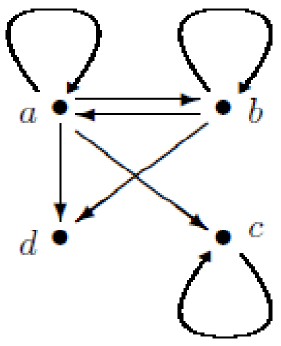
Representing graphs

By a Matrix

Number of edges going from vertex i to vertex j.



	x	y	w	z
x	1	1	0	2
y	1	0	0	0
w	0	0	0	0
z	2	0	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

Paths in simple graphs

Path: a sequence of vertices travelling along edges.

Length of a path is the number of edges in it.

Simple path: path which does not contain the same edge twice.

Hamiltonian path: a simple path passing through every vertex exactly once.

Cycles in simple graphs

Cycle: a path beginning and ending at the same vertex.

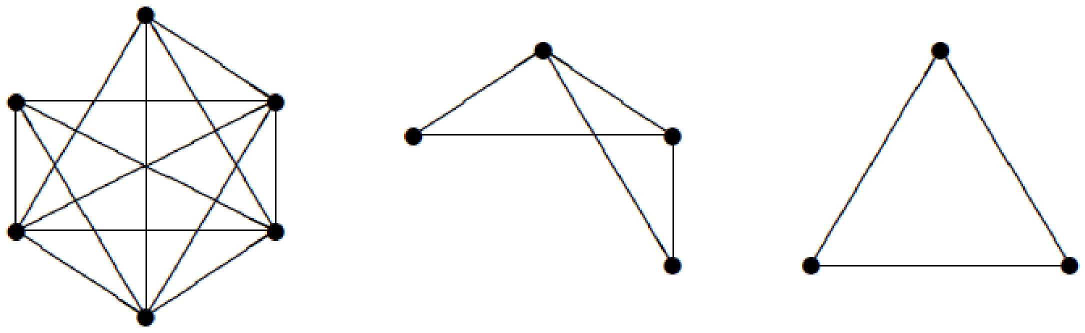
Length: the length of a cycle is the number of edges in it.

A cycle is **simple** if it doesn't contain the same edge twice.

Hamiltonian cycle: a simple cycle passing through every vertex exactly once.

Subgraphs

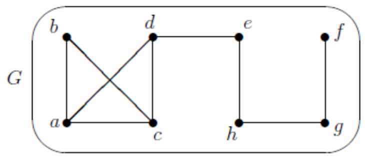
We can make subgraphs by eliminating vertices.



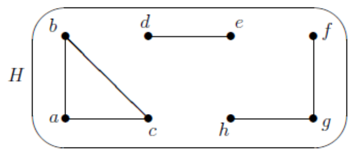
is a subgraph of K_6.

Connected or disconnected simple graphs

A grapg is **connected** if there is a path between every pair of distinct edges.



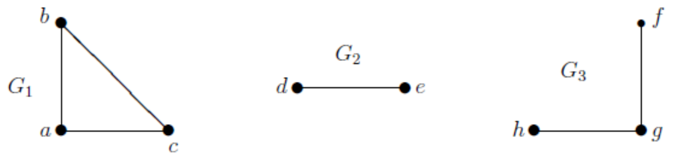
is connected



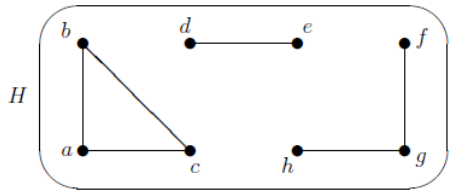
is not connected

A **connected compontent** 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:

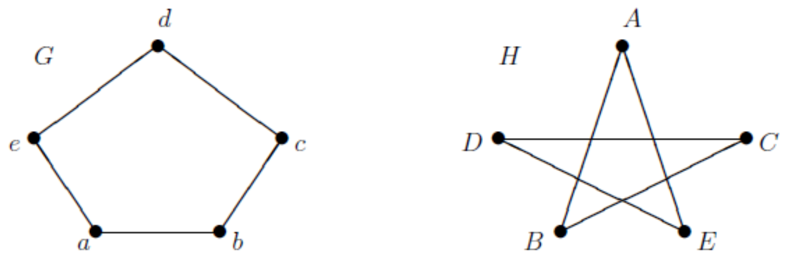


are the 3 connected components of



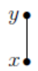
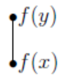
Isomorphism of graphs

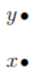
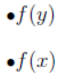
Isomorphic graphs are those that are the same but are drawn / seen different.



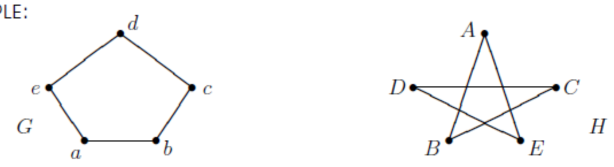
Two graphs are isomorphic if an **isomorphism** exists between the two:

- 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

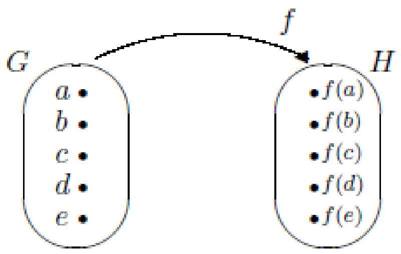
$$f(a) = A, \quad f(b) = B, \quad f(c) = C, \quad f(d) = D, \quad f(e) = E$$

is an isomorphism, showing that graphs G and H are isomorphic.

Invariants

A property *P* of a graph is called an **invariant**. If graph G has an invariant so does graph H.

- f is a <u>*bijection*</u> from the vertices of G (domain) to the vertices of H (codomain).



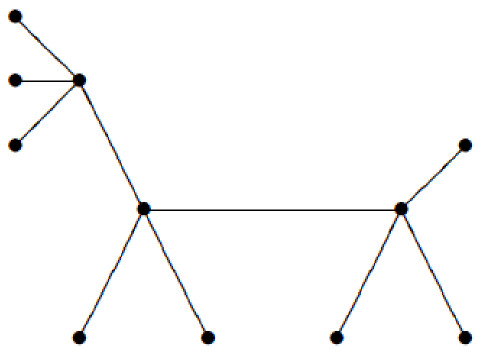
- As f is <u>*one-to-one,*</u> H has <u>at least 5</u> vertices.
- And as f is <u>*onto,*</u> H has <u>at most 5</u> vertices.

Isomorphic graphs have (**invariants**):

- The same number of vertices.
- The same degree for each vertex.
- Their adjacent vertex / verticies have the same degree.
- Contains a certain cycle.

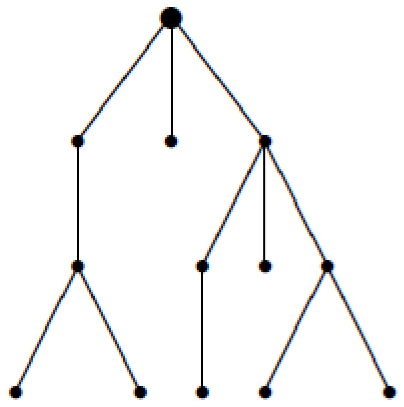
Special graphs: trees

Tree: a connected graph with no simple cycles.



Rooted tree

Rooted tree: a tree in which one vertex is defined as the root. To change an unrooted tree onto a rooted one, you can choose any vertex as the root and we normally write it at the top.



Isomorphic rooted trees: rooted trees are isomorphic if there's a bijection between their vertices such that **takes root to root**, edges to edges and non-edges to non-edges.

Basic Terminology

○ There is a parent, a child and siblings in order (siblings having the same parent).

- A childless vertex is a **leaf** (bottom left of the diagram).
- Vertices with at least one child are called internal.
- The **ancestors** of a point are all points that create a simple path to the root.
- The **descendants** are the ones that have the designated point as an ancestor.

Features of the rooted tree

- The **depth** or **level** of a vertex is the length of the unique path from a point to the root.
- The **height** of a tree is the maximum depth of its vertices.

We can create a **subtree** of a rooted tree by getting one point and separating it from the rest, all the points' descendants and all edges incident are part of the new tree.

Special trees

- A rooted tree is called a **m-ary tree** if every internal vertex has no more than m children.
- A rooted tree is called a **full m-ary tree** if all internal vertex has **exactly** m children.
- A rooted tree is called a **full binary tree** if every internal vertex has **exactly two children:** a left and a right child.

Counting vertices and edges of trees

- A **full binary tree** with n internal vertices contains $2n + 1$ vertices all together.
  - All internal vertices (except for the root) have two children hence there is 2n vertices other than the root.
- A full **m-ary** tree with n internal vertices has $m \cdot n + 1$ vertices.
- A tree with m vertices has $m - 1$ edges.

- It can be proved by induction on m.

Binary search trees

Binary search trees are good to represent elements in a linearly ordered list.

Linearly ordered list: a sequence whose elements are linearly ordered (not necessary in order of listing).

FOR EXAMPLE:

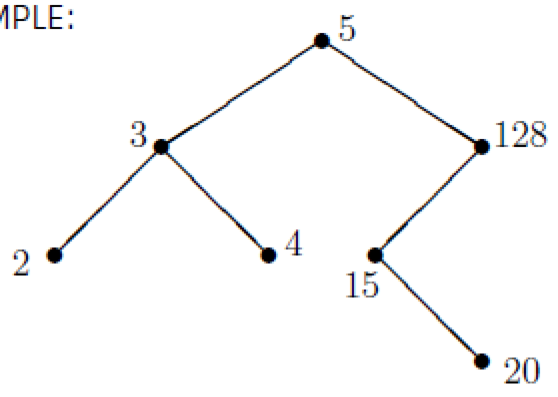
- $(5, 128, 3, 2, 15, 4, 20)$ is a list of natural numbers, natural numbers can be ordered by the $\leq$ relation
- (*mathematics, physics, geography, geology, psychology*) is a list of words, words can be ordered by the *lexicographical order relation* $\prec$ (see next slide)

NOTE: lexicographical order relation uses ordered letters to order words.

A symbol B, etc.

Example of linear order for greater than or equal to $\leq$ from the list $(5, 128, 3, 2, 15, 4, 20)$.

FOR EXAMPLE:



Building Binary Search Trees

1. Put all members in linear order from left to right.
2. The first item is assigned as the **root**.
3. **Comparing**: we take the next item of the list and compare it to the *old* vertex. If is **less than** the one before the old vertex has a left child. If it's **more than** then we add a right child.

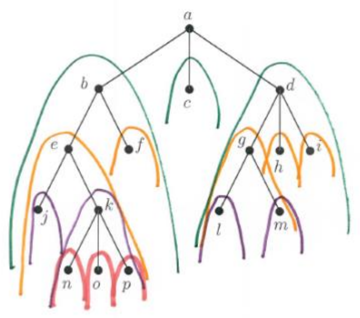
4. **Adding**: when the new item is **less than** the old vertex and the original one we insert a new left child and label it with the new item. For the **larger than** the same thing occurs with the exception that it creates a right child.

Tree traversal

Rooted trees are used to store information. We need systematic procedures for visiting each vertex of a rooted tree to access data, we call these procedures **traversal algorithms**.

- **Preorder traversal:** visit the root, then continue traversing the subtrees in order, from left to right.
- **Inorder traversal:** begin travesing the leftmost subtree in order, then visit the root, then continue traversing subtrees in order, from left to right.
- **Postorder traversal:** begin traversing leftmost subtree in postorder, then continue traversing subtrees in postorder from left to right, finally visit the root.

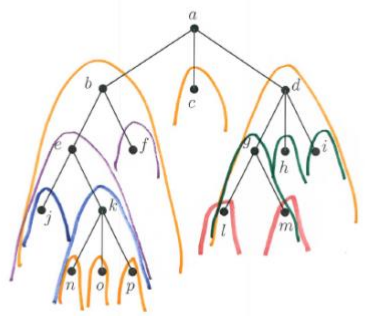
Preorder traversal



Visit the root, then continue traversing subtrees in preorder, from left to right:

$a, b, e, j, k, n, o, p, f, c, d, g, l, m, h, i$

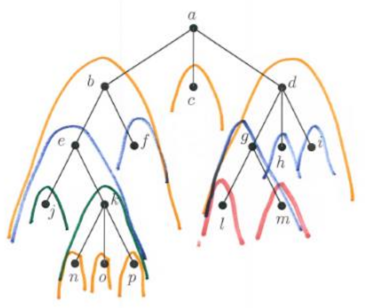
Inorder traversal



Begin traversing leftmost subtree in inorder, then visit root, then continue traversing subtrees in inorder, from left to right:

$j, e, n, k, o, p, b, f, a, c, l, g, m, d, h, i$

Postorder traversal



Begin traversing leftmost subtree in postorder, then continue traversing subtrees in postorder, from left to right, finally visit root:

$j, n, o, p, k, e, f, b, c, l, m, g, h, i, d, a$

Finite Automata

Finite automata aka final state.

Basics

Alphabet: a finite set of symbols.

Word or string: over an alphabet is a final sequence of symbols written without commas.

Tortoise, awaseas, 11100011011, 10011155

Length of a word: the length of a word $|w|$ is the number of symbols in w.

The **empty word** (ε) is a word over any alphabet (but is never a symbol of any of our alphabets.

Concatenation of words: when a certain sequence of symbols is repeated a certain amount of time.

- $w^n = \overbrace{ww\ldots w}^{n}$

  FOR EXAMPLE: $(\heartsuit\square)^0 = \varepsilon$, $(01)^3 = 010101$, $a^4 = aaaa$
- $x\varepsilon = \varepsilon x = x$, for every word x

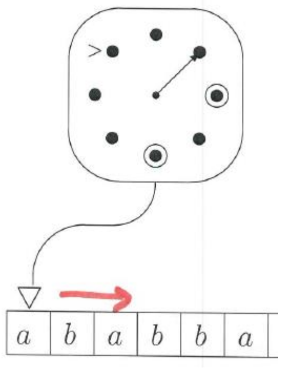
- If $w = xy$ then x is a prefix of w, and y a suffix of w.

  FOR EXAMPLE: tor is a prefix and se is a suffix of $tortoise$

  $tortoise$ is **both** a prefix and a suffix of $tortoise$

Language: a language is a set of words over an alphabet.

Finite automaton

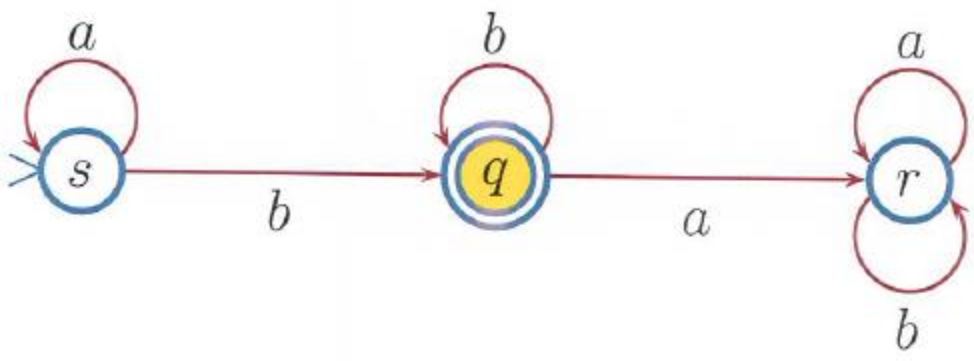


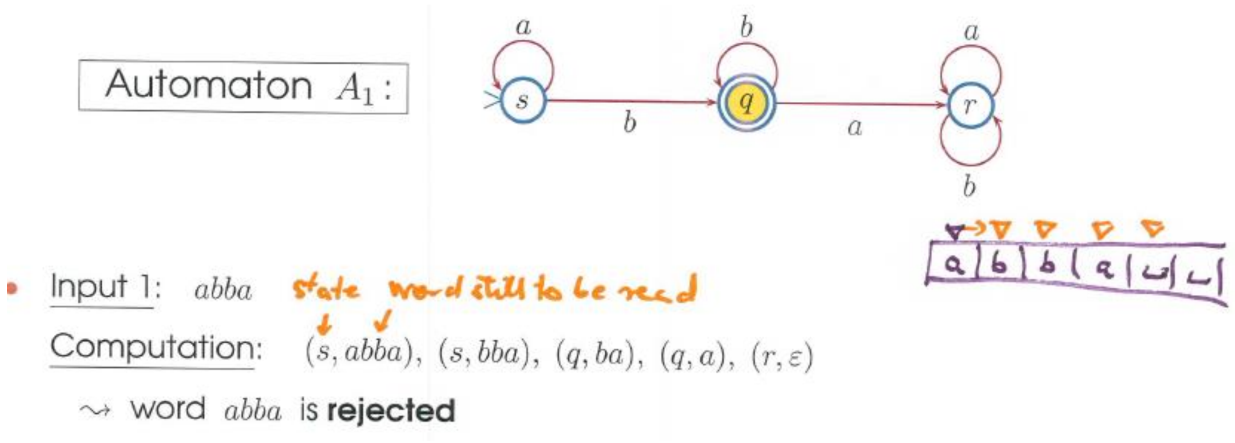
A finite automaton uses **constant memory** regardless of the input form.

Finite control device: a machine in which at any moment can be in one of its **states.** It is hardwired how it changed from one step to the next. There is an **initial state**, and possible **favorable states.** This device uses a reading head which reads an input tape which is divided in cells from left to right, each containing a character of the input alphabet. The automaton reads the rape and depending on the symbol of the tape it heads to another state. If when the automaton finishes it ends in a favorable state (starting from the initial state) then the inputted word is **accepted**, if not it's **rejected**.

Visual representation (state transition diagrams)

We can represent a hardwired automaton by a directed multigraph in which vertices represent states, arrow-edges indicate where to go with alphabet symbols, initial state is marked with a > and favorable states are double-circled.





Automaton A_1:

- Input 1: $abba$ state word still to be read

  Computation: $(s, abba)$, (s, bba), (q, ba), (q, a), (r, ε)

  $\rightsquigarrow$ word $abba$ is **rejected**

NOTE THE CORRECT NOTATION UP TOP.

Deterministic Finite Automaton DFA

To describe a **Deterministic Finite Automaton,** you need 5 things:

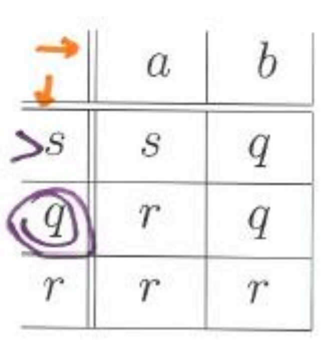
- Its states
- Its input alphabet
- Its initial state
- Its favorable states
- Its transition functions

A **word** is accepted by a DFA if the computation of the word ends in a favourable state, otherwise it's rejected.

It's called **deterministic** because there is a unique arrow for each symbol from one state to another.

Transition Tables

Another way of showing a DFA is by using a transition table.

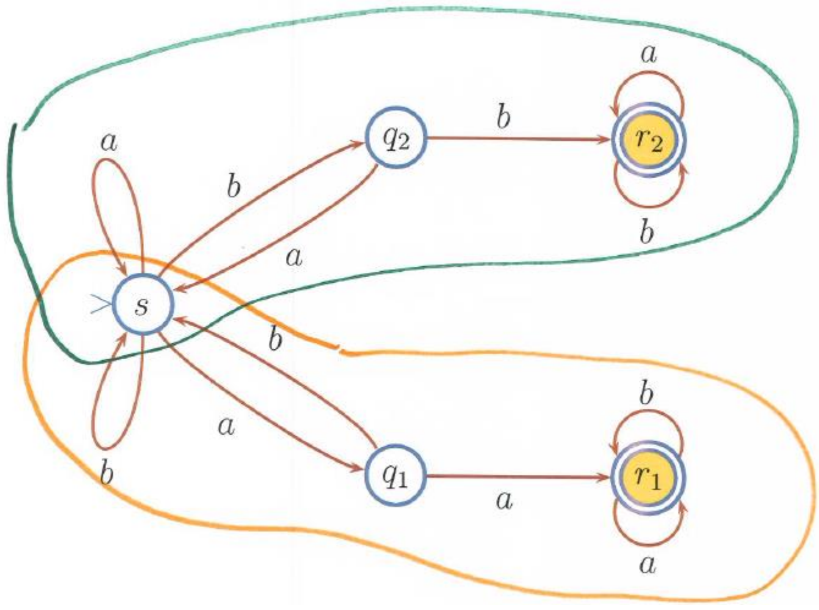


	a	b
$>s$	s	q
q	r	q
r	r	r

This would be the equivalent of the state transition diagram for A1.

Language of a DFA: $L(A)$ = the set of all words over its alphabet that the DFA accepts.

Whenever we combine two automatons, we no longer have a deterministic one, this is because given a certain input it can go to one place or another.

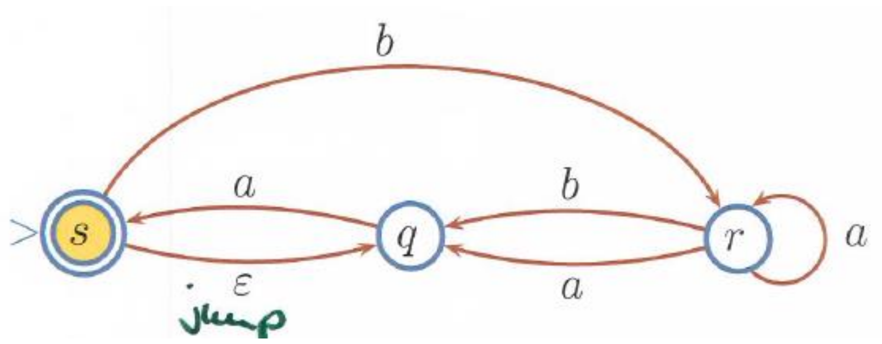


As we can see, there is more than one a-arrow leaving from the initial state s and more than one b-arrow. It's no longer determined to which state it will go to.

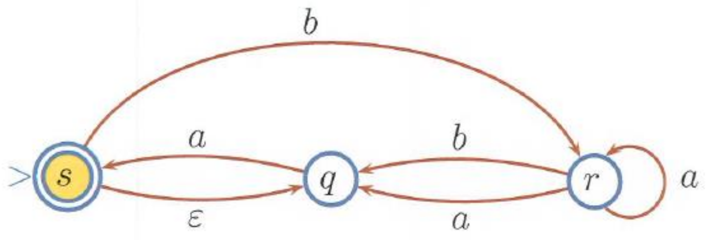
Nondeterministic Finite Automata

In nondeterministic finite automata the following can occur:

- There is more than one arrow for each symbol leaving a state (or arriving to a state).
- An automaton can get **stuck**, meaning it gets to a point in which there is no arrow for the symbol that's next.
- The machine is allowed to **jump** from a state to another (which will be symbolized with the empty word).



In an NFA there can be **more than one** computation on an input word. Some may end up on a favorable state and some will be rejected. That's why we say that a word is accepted by a NFA if **there is a computation that ends up in a favorable state** (all symbols must have been read).



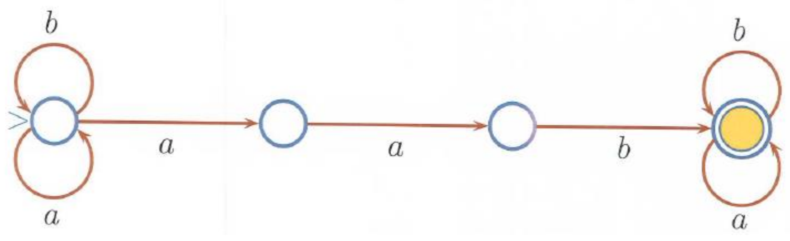
- Computations on input $baba$:

  $(s, baba), (q, baba)$ (stuck)

  $(s, baba), (r, aba), (r, ba), (q, a), (s, \varepsilon)$ $\rightsquigarrow$ $\boxed{\text{accepted}}$ . . .

Creating NFA's is much more simple than DFA's but they can be converted. For instance:

Task: *Design a finite automaton A such that $L(A)$ consists of* **all** *the strings of as and bs that contain $\boxed{aab}$ as a substring.*



We only have to make sure that one computation is accepted.

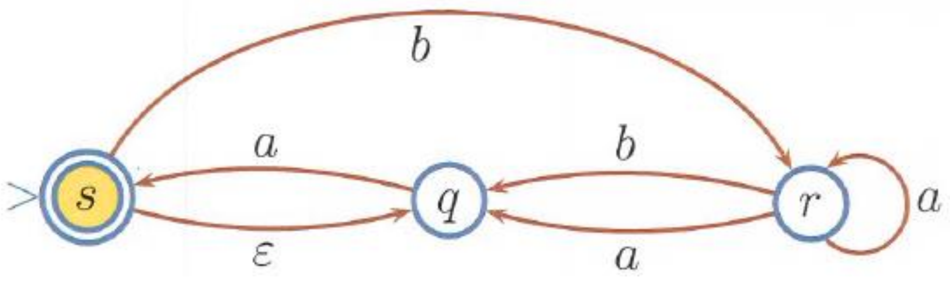
For every NFA A, there is a DFA A’ equivalent to A.

The Subset Construction

Is a method which turns a NFA into its equivalent DFA.
Given a **NFA** we will need new things to define:

- New states
- New initial state
- New favorable states
- New transitions

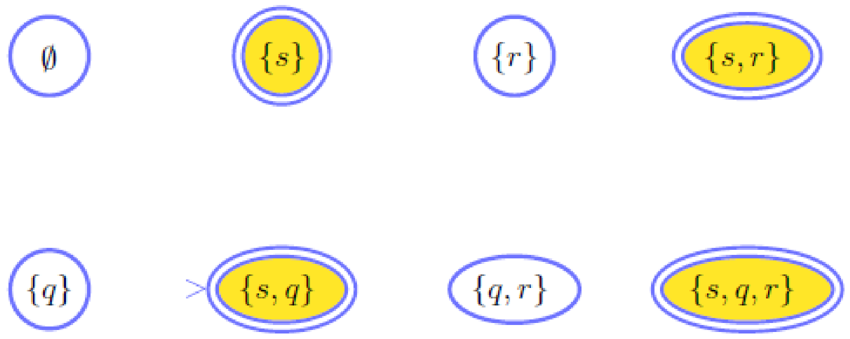
NFA:



The new states

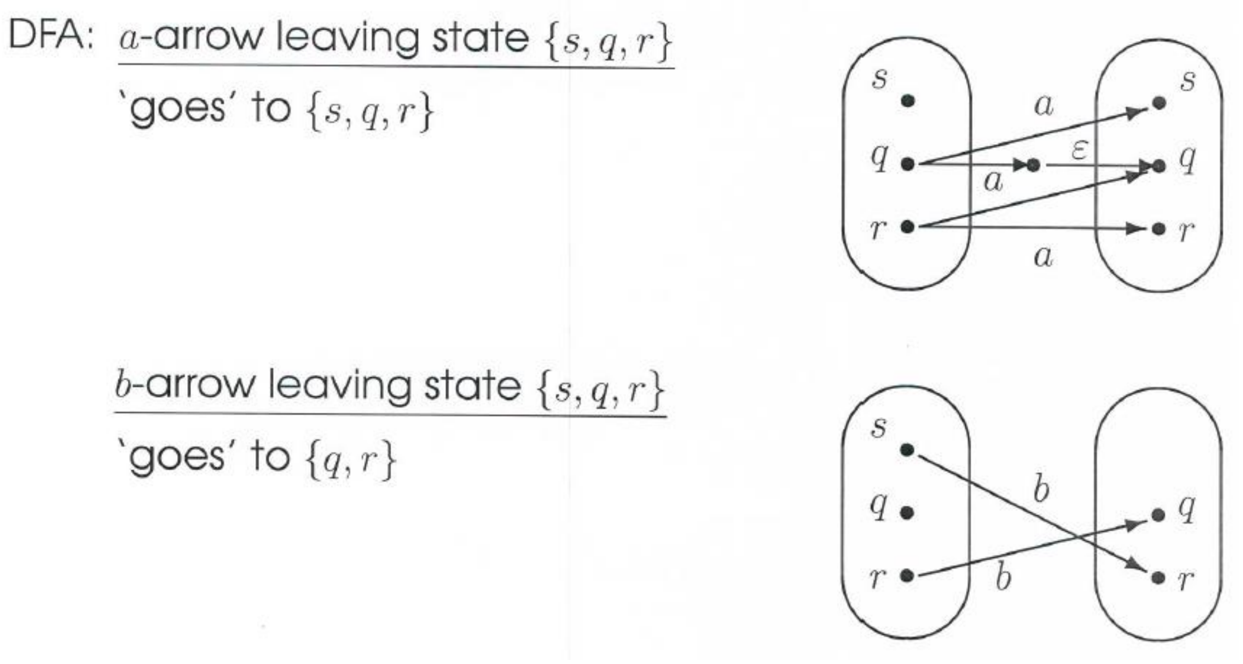
New states: are all the subsets of the NFA’s states.
New initial state: the set containing the NFA’s initial states plus all those states that are reachable from it by some jumps.
New favorable states: those subsets that contain at least one of the NFA’s favorable states.

$$\{s, q, r\} = 2^3 = 8\ subsets$$

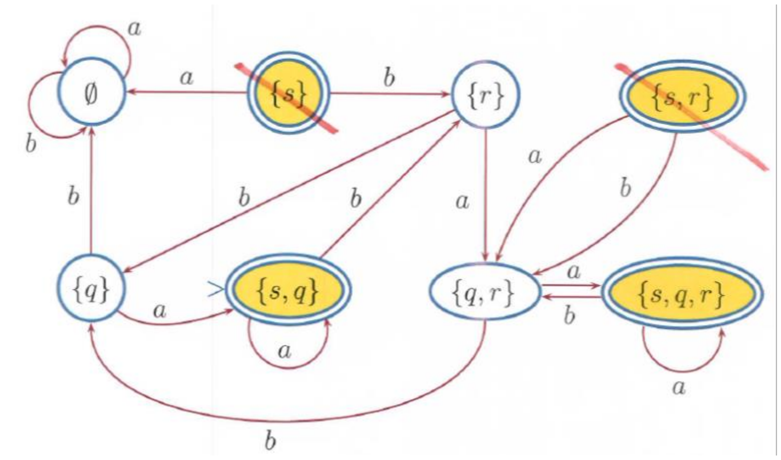


The new transitions

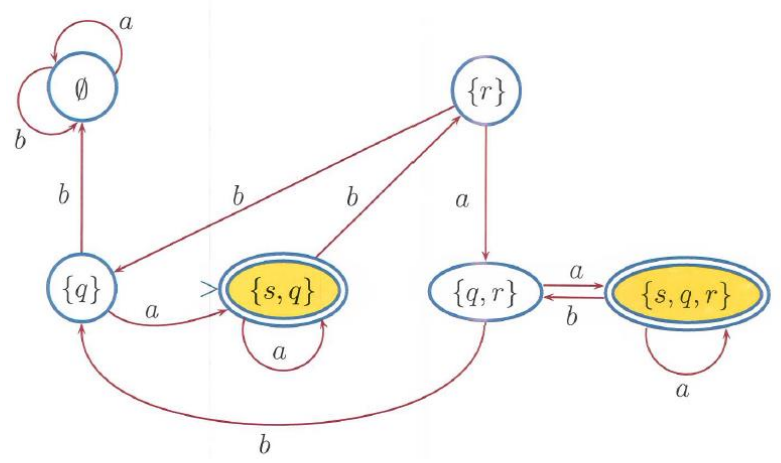
We create diagrams for each symbol in our alphabet which tells us how to get from a state to another including jumps (but not when the jump occurs first and then the symbol).


DFA: a-arrow leaving state $\{s,q,r\}$
'goes' to $\{s,q,r\}$
b-arrow leaving state $\{s,q,r\}$
'goes' to $\{q,r\}$
s
q
r
a
ε
b

As we see, **all q's must have an a-arrow towards a and another q, etc.**



Then we eliminate all the unreachable states… (bearing in mind that we can never delete the initial state).



Thus, we can create the new transition table:

	a	b
$\{s\}$	$\emptyset$	$\{r\}$
$\{q\}$	$\{s,q\}$	$\emptyset$
$\{r\}$	$\{q,r\}$	$\{q\}$
$\{s,q\}$	$\{s,q\}$	$\{r\}$
$\{s,r\}$	$\{q,r\}$	$\{q,r\}$
$\{q,r\}$	$\{s,q,r\}$	$\{q\}$
$\{s,q,r\}$	$\{s,q,r\}$	$\{q,r\}$
$\emptyset$	$\emptyset$	$\emptyset$

Regular Expressions

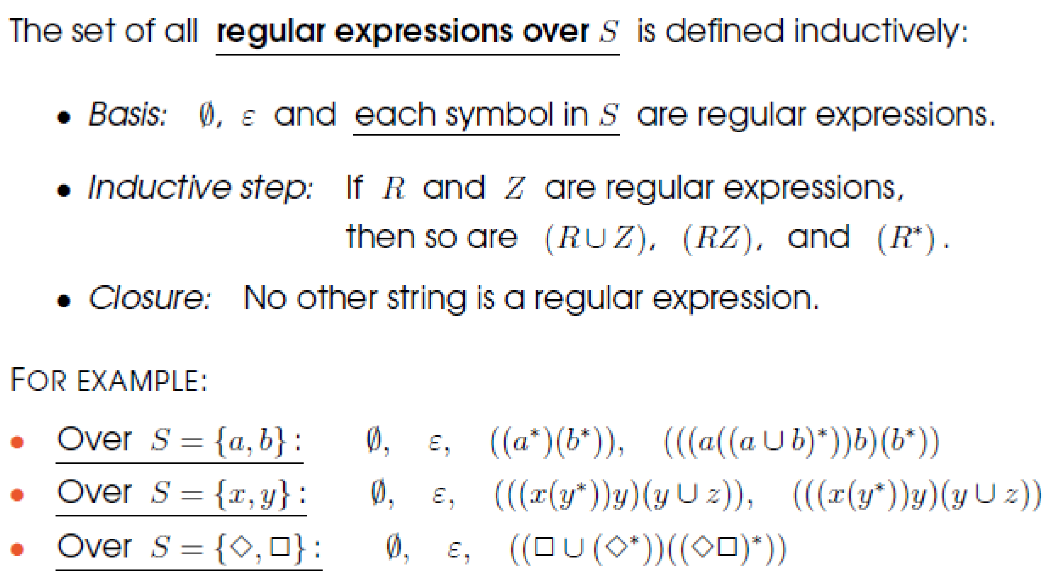
A regular expression is a way to describe an **infinite language** with a **finite "pattern"**.
Consider the following language: a string of 1's and 0's which contain two or three occurrences of 1's, in which the first and the second are not consecutive.

NOTE: $something^*$ means repeated * times, which can also be zero.

- The string can start with any number of zeros: 0^*
- Then comes the first one: 0^*1
- It has to be alone so a zero must follow: 0^*10
- Then any possible number of zeros can follow: 0^*100^*
- Then comes the second 1: 0^*100^*1
- Then either no more 1's but as many 0's as you want or a third 1 followed by any number of 0's: $0^*100^*(0^* \cup 0^*10^*)$

This final expression is said to be the **regular expression** representing the Language.

Describing regular expressions by induction


The set of all regular expressions over S is defined inductively:

- *Basis:* $\emptyset$, ε and each symbol in S are regular expressions.
- *Inductive step:* If R and Z are regular expressions, then so are $(R \cup Z)$, (RZ), and (R^*).
- *Closure:* No other string is a regular expression.

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, $((\Box \cup (\Diamond^*))((\Diamond\Box)^*))$

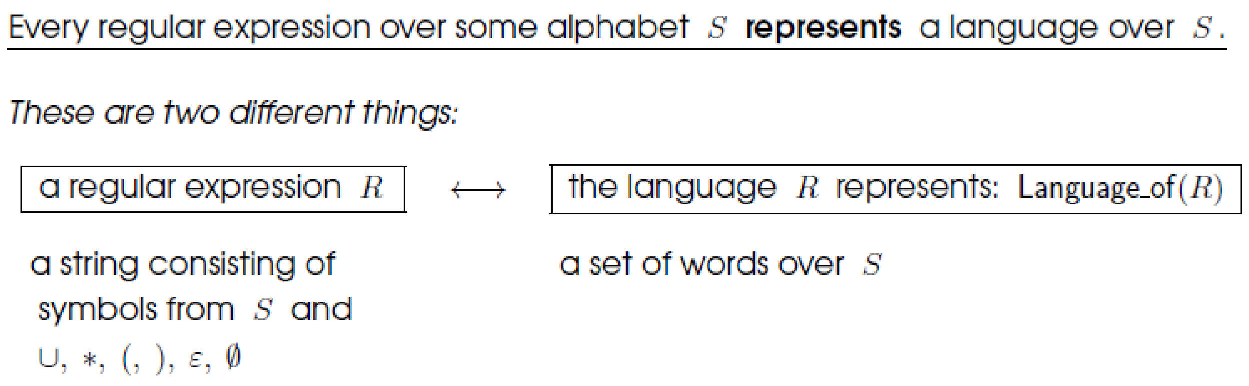
Conventions

- We normally omit the outermost brackets.

- Omit brackets $ababb$ instead of $(((ab)(ab))b)$
- * binds tighter than concatenation. aab^* *instead of* $(aa)(b^*)$.

NOTE: $(aa)^*b$ and aa^*b are not the same.

Summary



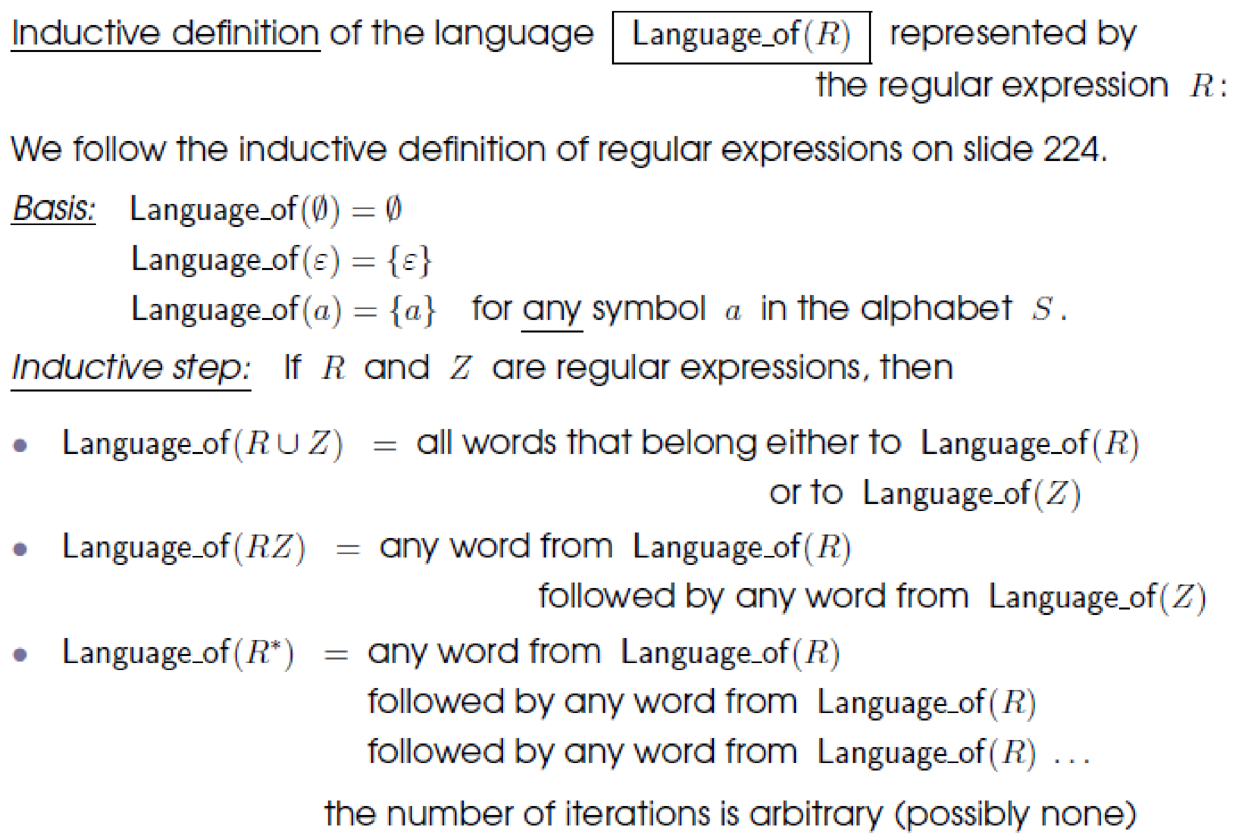
Every regular expression over some alphabet S **represents** a language over S.

These are two different things:

$\boxed{\text{a regular expression } R}$ $\longleftrightarrow$ $\boxed{\text{the language } R \text{ represents: Language\_of}(R)}$

a string consisting of symbols from S and $\cup$, $*$, (,), ε, $\emptyset$

a set of words over S



Inductive definition of the language $\boxed{\text{Language\_of}(R)}$ represented by the regular expression R:

We follow the inductive definition of regular expressions on slide 224.

Basis: $\text{Language\_of}(\emptyset) = \emptyset$

$\text{Language\_of}(\varepsilon) = \{\varepsilon\}$

$\text{Language\_of}(a) = \{a\}$ for any symbol a in the alphabet S.

Inductive step: 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 $\text{Language\_of}(R)$ ...

  the number of iterations is arbitrary (possibly none)

Regular Languages

Regular language: any language that can be represented by a regular expression.

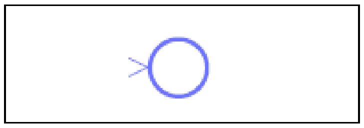
Not every language is regular.

Regular expressions to NFA

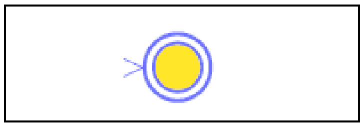
The procedure which converts any regular expression R to an NFA A_R such that $L(A_R) = Language\_of(R)$ operates along the inductive definition of the regular expression R.

Basis:

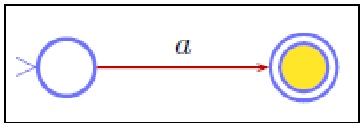
- If $R = \emptyset$. Then Language_of$(R) = \emptyset$. A_R :



- If $R = \varepsilon$. Then Language_of$(R) = \{\varepsilon\}$. A_R :



- If $R = a$. Then Language_of$(R) = \{a\}$. A_R :



Example:

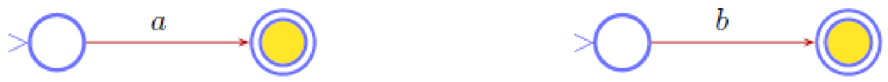
We apply the procedure to the regular expression

$$((a \cup ab)^* ba)^*$$

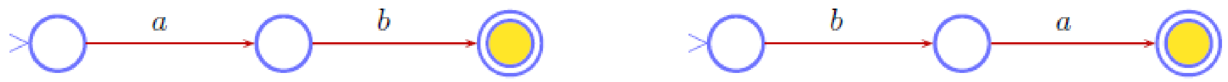
We construct step by step (going 'inside out') an NFA A such that

$$L(A) = \text{Language\_of}(((a \cup ab)^* ba)^*)$$

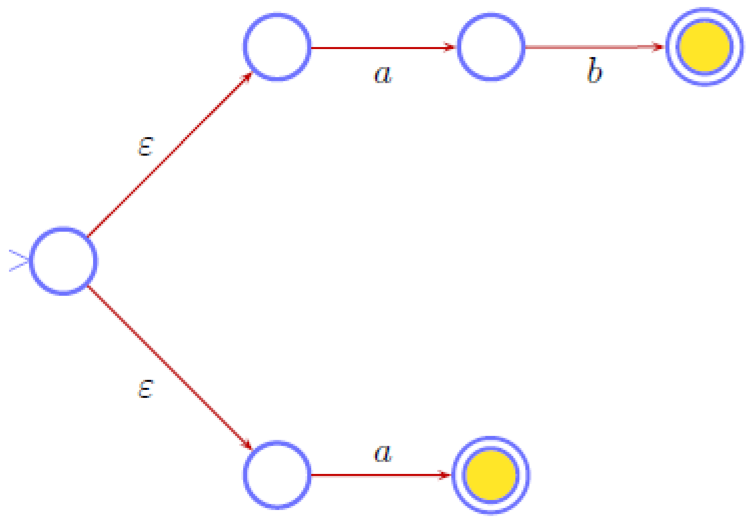
<u>**Step 0:**</u> automata accepting Language_of(a) and Language_of(b)



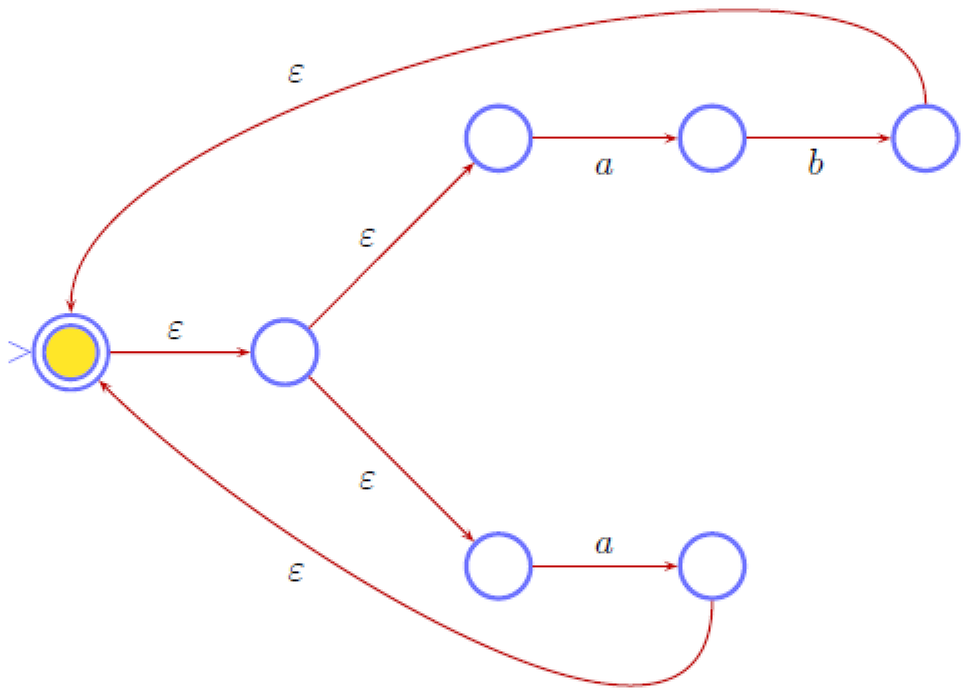
<u>**Step 1:**</u> automata accepting Language_of(ab) and Language_of(ba)



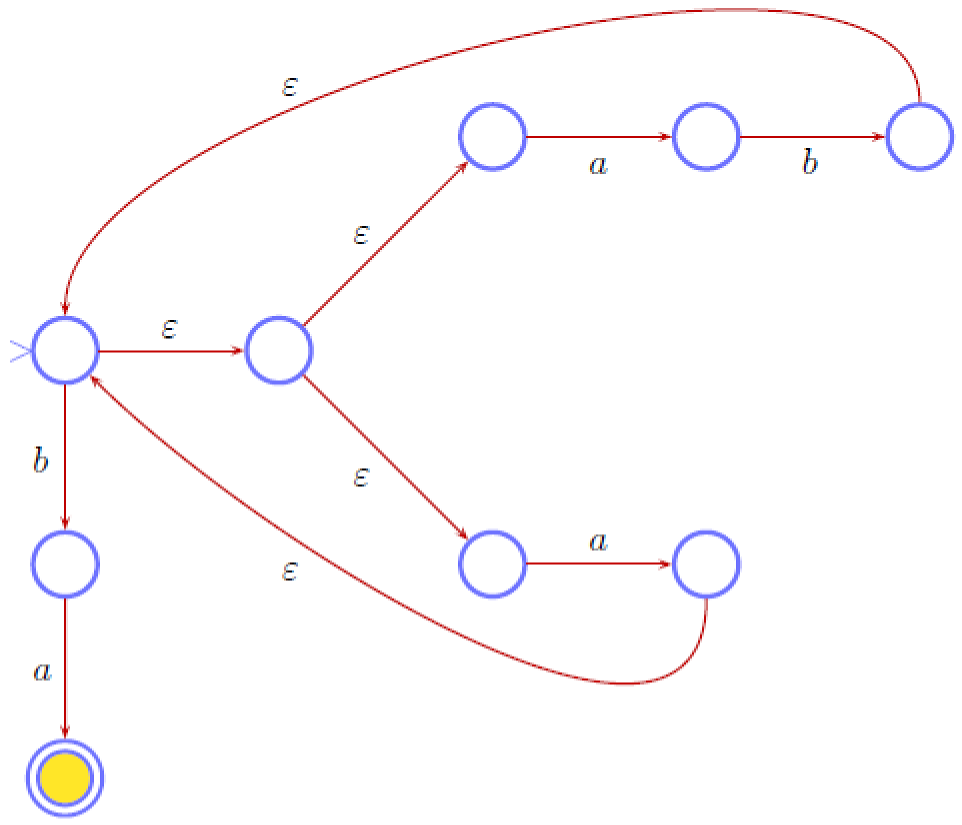
Step 2: automaton accepting Language_of$(a \cup ab)$



Step 3: automaton accepting Language_of$((a \cup ab)^*)$

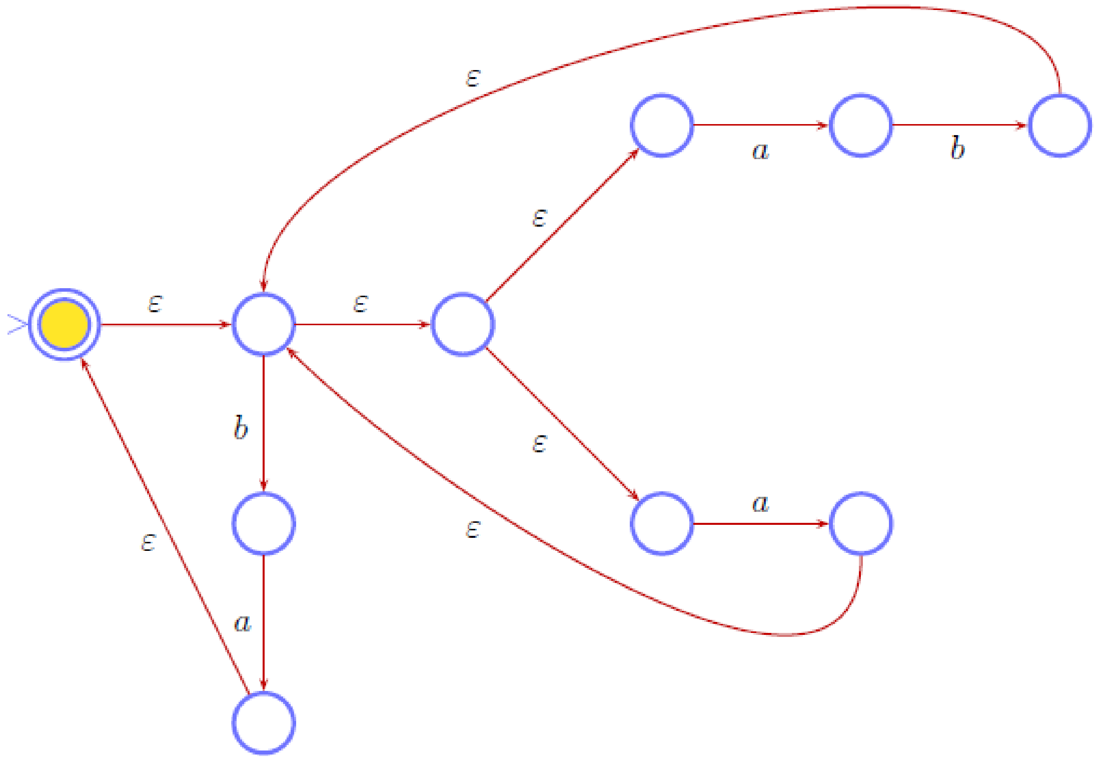


Step 4: automaton accepting Language_of$((a \cup ab)^* ba)$



Final step of the procedure

Automaton accepting Language_of$\left(\left((a \cup ab)^* ba\right)^*\right)$:



Summary

- Finite automata is said to use **fixed amounts of memory** (states) regardless of input.
- Finite automata can be used as **recognition devices:** accept certain inputs, and reject others.
- **Nondeterminism** does not increase the computational power of finite automata, but **nondeterministic** are easier to design than **deterministic** ones.
- Languages accepted by finite automata are the regular ones.

Since finite automata are theoretical models for programs using a constant amount of memory regardless of the input, one ought to expect that the power of such programs should be quite limited.

Nonregular languages

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 entire prefix a^n 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:

this language L is indeed **not regular.**