COMPUTER SCIENCE FOR THE BUSY DEVELOPER

A Relatively Quick Overview
Of Core Concepts Of Theoretical Computer Science

DRAFT

Rakhim Davletkaliyev

Codexpanse
Helsinki, Finland

2019

Contents

Part I

Intro

This course and the book constitute an high speed overview of the most important fundamental computer science areas. It is intended for intermediate and professional developers who, for any reason, are interested in getting to know the formal, academic side of CS better.

The goal is to provide an overview deep enough so that you end up understanding the ares, their problems and the connections between them. And shallow enough so that you aren't buried under hundreds of pages of proofs, formalizations and exercises.

We will start by trying to understand what computer science is and why it isn't a new area at all. We'll consider the computability as a fundamental property of reality rather than a technological apparatus.

We shall then proceed to learning essential mathematics necessary for further topics. These include proof techniques, notation and logic.

Next, we will learn about the following topics:

1. Set theory.
2. Algorithms. Complexity and examples of important algorithms in sorting.
3. Abstract Data Types.
4. Graph theory.
5. Theory of computation.
6. Cryptography.
7. Information theory.

Since it seems impossible or at least unpractical to put all of computer science curriculum into a single course, even in a minified format, we shall leave the following topics for the last chapter. In it, we will give an overview to them:

- Abstract algebra
- Category theory
- Type theory
- Computational geometry

Part II

Foundations of Math

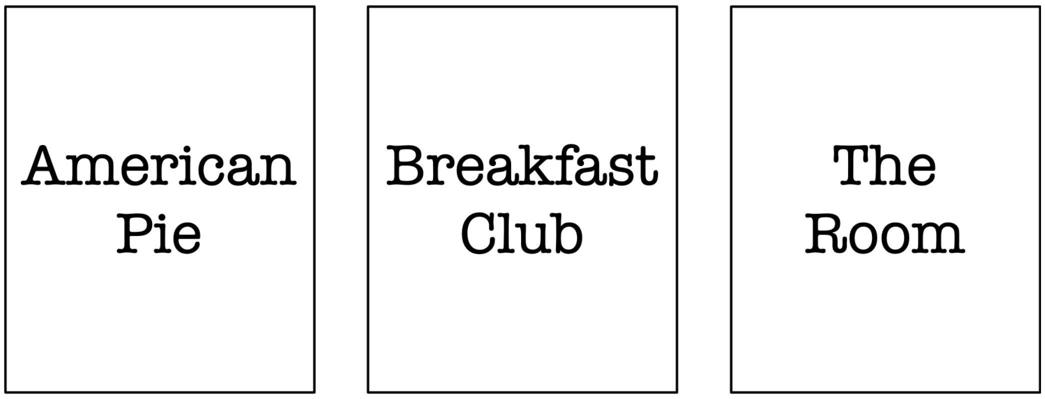
Chapter 1

Set theory

1.1 Intro to sets

Set theory is an important area of math which lays as a foundation of many computer science topics, such as databases and types in programming. Set theory isn't difficult conceptually, and is generally nice to think about, especially if you like to visualize.

A **set** is simply a collection of things.



Elements of set A.

A thing can be anything: name, object, idea. It's a very abstract notion. For example, I can define a set of movies I have seen: American Pie, Breakfast Club, The Room. Yes, I have only seen those three movies in my entire life. I am busy writing texts about set theory.

Actually, I've watched The Room five times. Does this affect the set? No, because it doesn't change the notion of "what movies I have watched". This obvious idea is an integral property of sets: they contain only unique objects.

Nothing stops us from defining a set of movie viewings, though. In that case, each element would be a particular instance of a "movie watching event", and, in my sad case, would contain 7 elements.

If an object o is in set A, we say o is a member of A. To express this fact easily, mathematicians use the following notation: $o \in A$.

So, if $A =$ Movies I've watched and $o =$ The Room, then $o \in A$. But if $b =$ Avengers, then $b \notin A$. Now you can generate infinitely many "nerdy" T-shirts with prints like joy $\in$ mylife or party $\in$ dahouse.

If we were to look inside set A, it might look like this:

$$\{s, m, o\} \tag{1.1}$$

Sets don't have the notion of order, so it doesn't matter how you mention its members as long as you mention them all. So, all these:

- $\{s, m, o\}$
- $\{s, o, m\}$
- $\{m, s, o\}$
- etc.

describe the same set A.

We are lucky: set A is finite and quite small. But sets can be infinite, and it would be impossible to write down all of its members. There are ways to describe such sets nevertheless. For example, the set of all natural numbers from 1 to n can be described like so:

$$\{1, 2, 3, 4, ...n\} \tag{1.2}$$

Here we rely on reader's common sense and assume he or she can rightfully deduce what is meant by the ellipsis $(\ldots)$. Notations like this are very common in math, and while it isn't awfully strict, the idea is to be as unambiguous as possible. For example, when describing a set of odd natural numbers from 1 to n, this would be bad:

$$\{1, 3, ...n\} \tag{1.3}$$

simply because the sample isn't long enough to make a single assumption.

Another example of a set that exists in math is the set of Boolean values:

$$B = \{T, F\} \tag{1.4}$$

T and *F* stand for *True* and *False*. Certain tests (or, in other words, questions) that can be answered with "yes" and "no", produce values of said set. So, we can say that

$$(x > y) \in B, \text{where } x \in \mathbb{N} \text{ and } y \in \mathbb{N} \tag{1.5}$$

In other words, the answer to the question "is x greater than y" is a member of set B, as long as x and y are members of $\mathbb{N}$ (i.e. natural numbers).

Math in infinitely composable. Most programming languages can only dream of the level of composability and flexibility mathematicians enjoy. While not immediately useful, we could compose the following statements about sets:

$$((x + y) \in \mathbb{N}) \in B, \text{where } x, y \in \mathbb{N} \tag{1.6}$$

Here we said "when x and y are members of $\mathbb{N}$, then the answer to the question 'is $x + y$ a member of $\mathbb{N}$' is a member of B."

Many groups in mathematics are sets: numbers of different types (natural, irrational, complex, etc.), notions of different types (e.g. Boolean), solutions to particular problems (e.g. graphs that satisfy a certain property). But describing groups of elements is just the tip of the iceberg. Set theory, with its axioms and definitions, can play a role of a foundational area of math, from which many other areas can be derived.

In this course and the book we're not going to talk about how set theory (and category theory) can "generate" a large portion of the whole math. But these topics are among the most fascinating frontiers of modern mathematics.

1.2 Empty set

Mathematicians love zero. Ever since its inception around 1770 BC, zero is an important part of mathematical models. The notion of "nothingness", which zero reflects in the context of counting, is present in all areas. In the context of sets, nothingness is an *empty set*.

$$\varnothing = \{\} \tag{1.7}$$

Why would we need empty sets? Well, sometimes we want to describe a notion of having no objects under a certain description. For example, since I only watched 3 movies in my life, and all of them were American, I can describe:

$$\varnothing = \text{the set of all non-American movies I've watched.} \tag{1.8}$$

A more mathematical example would be something like this:

$$\varnothing = \{x | x \in \mathbb{N} \text{ and } x < 0\} \tag{1.9}$$

which says that the set of natural numbers smaller than zero is an empty set. It's a formal way to say that there are no natural numbers less than zero.

Note the use of vertical line `|`, it is a shorter way to say "where".

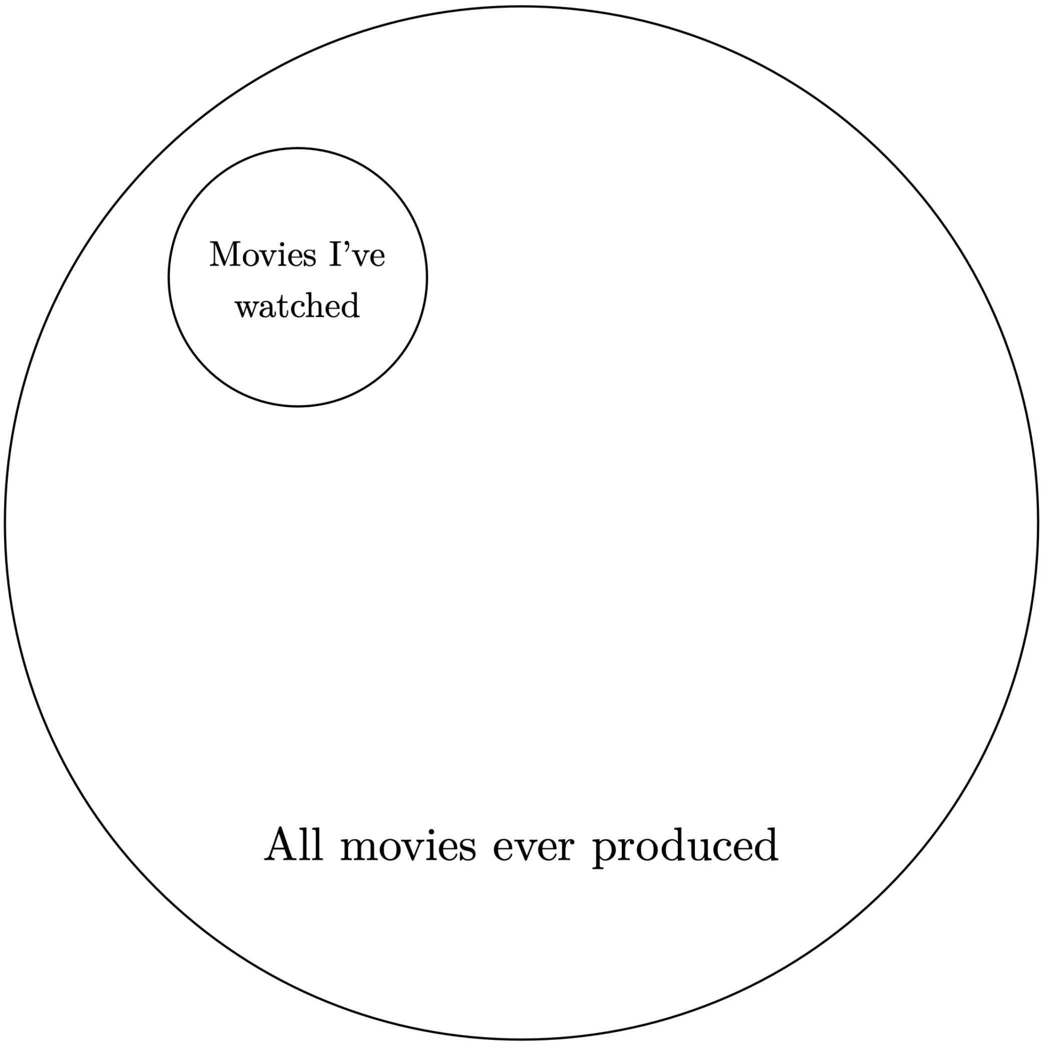
1.3 Subset, superset

When all members of set A are present in another set B, then A is a **subset** of B. Let's say set B is the set of all movies ever produced. Then A (movies I've watched) is clearly a subset of B. This notion is expressed like so:

$$A \subseteq B \tag{1.10}$$

To look at things from the other end, B is a **superset** of A:

$$B \supseteq A \tag{1.11}$$



A set and its subset.

You know what else is a subset of B? An empty set!

$$\varnothing \subseteq B \tag{1.12}$$

This either sounds absolutely natural to you or extremely weird. It makes perfect sense to a mathematician, because it's easy to argue: *all* members of $\varnothing$ are present in B, all zero of them.

It gets weirder. As per our definition, if all members of a set are also present in another set, then the one is a subset of the other. This means any set is a subset of itself.

$$A \subseteq A \qquad B \subseteq B \qquad Z \subseteq Z$$

(1.13)

By extension, if two sets are the same, then either of them is a subset of the other.

$$\text{if } A \subseteq B \text{ and } B \subseteq A \text{ then } A = B \tag{1.14}$$

When we look at a statement $A \subseteq B$, we often need to know whether $A = B$ or not. To distinguish between the two cases, mathematicians use a special notion of a **proper subset**.

If $A \in B$, but $A \neq B$, then A is a proper subset of B.

$$A \subset B \tag{1.15}$$

Since I haven't watched all the movies ever produced, I can say that A is a proper subset of B. So, a set is a subset of itself, but is never a proper subset of itself.

1.4 Cardinality

Before we start to talk about sizes and products, let's first introduce a new notion: **sequence**. A sequence is simply a series of elements. Unlike sets, sequences have order and may contain duplicate values. In this sense sequences are more down to earth, practical collections.

Sequences are written with round brackets. For example, a sequence of prime numbers in ascending order is:

$$(2, 3, 5, 7, 11, ...) \tag{1.16}$$

Note that it is an infinite sequence. An example of a finite sequence is a sequence of natural even numbers smaller than 10:

$$(1, 2, 3, 4, 5, 6, 7, 8, 9) \tag{1.17}$$

If A is a finite set (that is, A is not infinite, and we can count how many elements there are in it), we use $|A|$ to denote the number of elements of A. This is the **cardinality**. For example:

$$|\{4, 8, 15, 16, 23, 42\}| = 6, \qquad |\varnothing| = 0, \qquad |\{\{a, e\}\}| = 1.$$

(1.18)

Note the last example: it defines a set which contains one set, thus its cardinality is 1. The number of elements of the internal set is irrelevant. Programmers find this obvious, as it reminds them of nested data structures like arrays of arrays.

Knowing about sets, subsets and cardinalities, let's look at a statement and prove it.

Statement 1: *If A is a finite set of m elements, then there are 2^m subsets of A.*

Proof: Suppose $A = \{a_1, a_2, ..., a_m\}$ and PA is the set of all subsets of A. Then we can divide PA into $2 = 2^1$ collections: subsets of A which contain a_1 and those which don't. Considering the next element a_2, we get 4=2^2 collections:

1. One which contains both a_1 and a_2,
2. one which contains a_1, but not a_2,
3. one which contains a_2, but not a_1,
4. one which contains neither a_1 nor a_2.

Continuing this way, we see that there are 2^m subsets of A, each subset is determined by the fact of whether or not each a_j is included, as j goes from 1 to m.

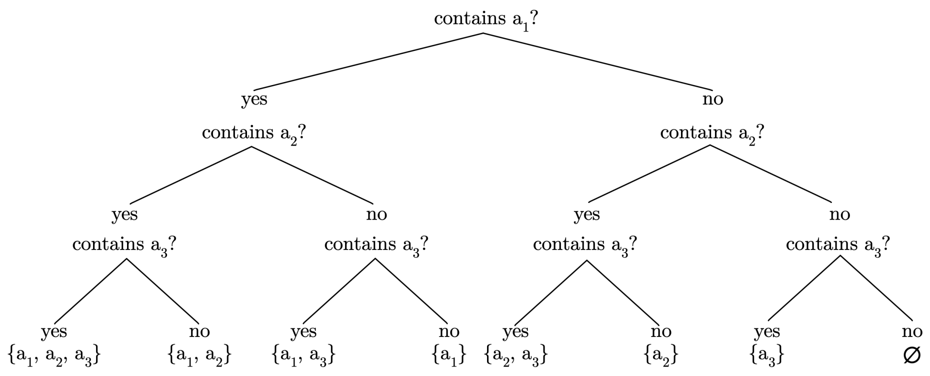
This sort of counting by inclusion / exclusion is a popular technique in math and especially in computer science. One can visualize such proof for a particular case by drawing a "binary decision tree".

Consider set $A = \{a_1, a_2, a_3\}$. The proof states that there are $2^3 = 8$ subsets of A. Do describe each imaginable set we need to determine whether each element is in it or not. Starting from a_1, we need to answer "yes" or "no", and proceed to ask the same question for a_2 and then for a_3.

1.5 Cartesian product

Given two sets A and B, we are often interested in all ordered pairs of their elements. For example, if $A = \{a, b\}$ and $B = \{1, 2\}$, the ordered pairs are as follows:

$$(a, 1) \qquad (a, 2) \qquad (b, 1) \qquad (b, 2)$$



(1.19)

The set of all such pairs is called the **Cartesian product** of A and B. The name comes from the French mathematician René Descartes.

Frans Hals, Portret van René Descartes

Formally, Cartesian product can be described like this:

$$A \times B = \{(a, b) | a \in A, b \in B\} \tag{1.20}$$

Descartes saw that the number plane x, y could be represented as a product of two sets of real numbers.

$$\mathbb{R} \times \mathbb{R} = \{(x, y) | x \in \mathbb{R}, y \in \mathbb{R}\} \tag{1.21}$$

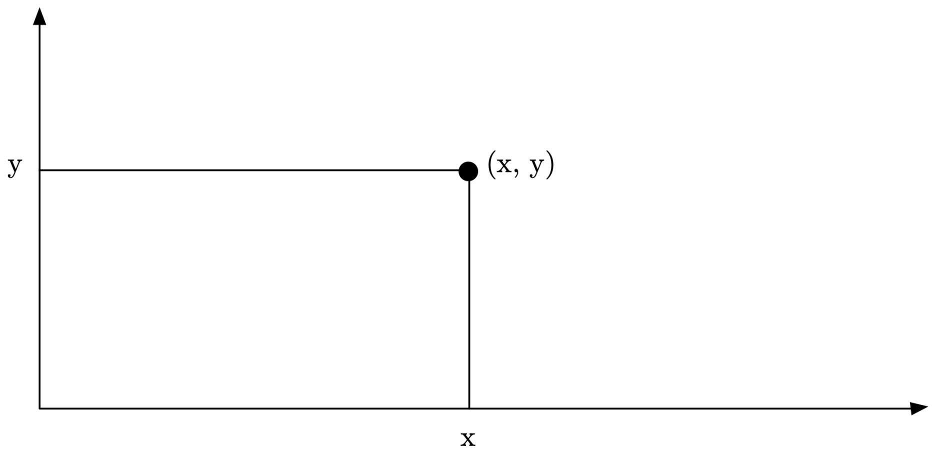


Figure 1.2: Each point on x, y plane is in the Cartesian product of 2 sets of real numbers.

1.6 Union

Sets on their own are a bit boring, frozen things. Interesting results can be observed when we consider interactions between sets. At the same time, these interactions can be viewed as frozen things themselves, not actions or processes.

This happens often in math: the same idea can be viewed as either an action or a thing. Even functions, seemingly action-able, moving notions, can (and will) be defined as mere static constructs. It's interesting to ponder about these things, draw analogies to physics and time.

Regardless, let's quickly overview three main sorts of interactions. Chances are they are already very familiar to you, especially if you've done any SQL.

Union of a collection of sets is the set of all elements of the collection.

Given two sets A and B, union is defined as:

$$A \cup B = \{x : x \in A \text{ or } x \in B\} \tag{1.22}$$

This reads as:

> Union of A and B is a set of elements x, where x belongs to A or x belongs to B.

For example, if $A = \{1, 2, 3\}$, and $B = \{3, 4, 5\}$, then:

$$A \cup B = \{1, 2, 3, 4, 5\} \tag{1.23}$$

Note that even though each set contains 3 elements, the resulting union contains 5 elements only. Since union is a set, all rules and properties regarding sets still apply, so it cannot contain element 3 twice.

It is sometimes necessary to keep duplicates somehow, without violating the rules of sets. One way is "tag" each element in both sets by generating a Cartesian product of each set.

$$A \times \{t_A\} = \{(1, t_A), (2, t_A), (3, t_A)\} \quad B \times \{t_B\} = \{(3, t_B), (4, t_B), (5, t_B)\}$$

(1.24)

So now instead of each number we deal with a sequence of two elements: the original number and a tag which refers to the original set. We use the term *n-tuple* for a sequence of length n. Thus, here we deal with sets of *2-tuples*.

Since all tuples are unique, the resulting union set contains 6 distinct elements:

$$A \times \{t_A\} \cup B \times \{t_B\} = \quad \{(1, t_A), (2, t_A), (3, t_A), (3, t_B), (4, t_B), (5, t_B)\}$$

(1.25)

Every time we learn about a new approach or an idea, your inner mathematician should aspire to generalize. We've already generalized the notion of union by providing a formal mathematical definition. Let us now define this so-called **disjoint union** $\sum A_i$ to be:

$$\bigcup_{1 \leq i \leq n} A_i \times \{i\} = \{(x, i) | x \in A_i \text{ and } 1 \leq i \leq n\} \tag{1.26}$$

It might seem like the notation is getting more and more complicated, but it's only a combination of existing notions and symbols, nothing new. The big U is a generalization of unions, here it is limited to all sets $A_i \times \{i\}$ where i goes from 1 to some n. You can think of i as a variable or a parameter. We see it is then used to define different sets (e.g. A_1, A_2, etc) and corresponding tagging sets $\{1\}$, $\{2\}$, etc. Then, on the right-hand side of the equation there's a set of many 2-tuples, each containing an element from A_i and a tagging number i.

To easily differentiate between regular unions and disjoint unions, we use two different symbols: $\cup$ for union, $+$ for disjoint union. Thus:

$$\bigcup_{1\leq i\leq n} A_i \times \{i\} = A_1 + ... + A_n. \tag{1.27}$$

Even if you haven't heard of set union, you've definitely seen Venn diagrams. They illustrate union, intersection and complement quite nicely.

`https://s3.amazonaws.com/thinkific/file_uploads/146581/images/8cf/e18/857/British_Isles_Venn_Diagram-en.svg.png`

Contrary to popular belief, Venn diagrams aren't suitable representations of `SQL JOIN`. You'll see why later.

When working with databases using SQL, union operation is a direct equivalent of union from set theory. The following example returns all usernames of both customers and managers:

```
SELECT username FROM users
UNION
SELECT username FROM managers;
```

This regular `UNION` operation behaves like set union in regards to duplicates. Thus, if both tables contain identical usernames, only one instance would end up in the union. An alternative SQL operator `UNION ALL` allows duplicates. The resulting collection of records is not necessarily a set, since it main contain identical records.

It is tempting to see SQL as set theory applied to databases, but it would be wrong to think this way. SQL can be considered a domain specific language for a particular application of relational algebra, which *incorporates* certain areas of set theory. In general, one can say that set theory and SQL *intersect.*

1.7 Intersection

Intersection of two sets A and B is the set containing all elements of A that also belong to B (or vice versa). In other words, it's a set of common elements.

$$A \cap B = \{x : x \in A \text{ and } x \in B\} \tag{1.28}$$

This reads as:

Intersection of A and B is the set of elements x, where x belongs to A and x belongs to B.

For example, if $A = \{1, 2, 3\}$, and $B = \{3, 4, 5\}$, then 3 is the only element present in both sets:

$$A \cap B = \{3\} \tag{1.29}$$

Sidenote: I quickly remembered which symbol – $\cup$ and $\cap$ – means which operation by noticing that $\cup$ looks like letter U, so it means union. I've had a similar moment when learning Boolean algebra and logic, where $\wedge$ means AND and looks like A without the horizontal bar.

SQL has `INTERSECT`:

```
SELECT username FROM customers
INTERSECT
SELECT username FROM managers;
```

Naturally, duplicates aren't an issue for intersection, because by definition one pair of elements results in one element, and not two. If there are two elements in one set, and one same element in the other set, the result is still one common element in the intersection.

A Venn diagram for intersection is often used to show commonality. For example, coming back to sets of favorite movies, we can compare your favorite movies and mine and see if there are any movies we both love. If no such movies exist, then our sets are **disjoint**. Formally, two sets are disjoint if their intersection is an empty set:

$$A \cap B = \varnothing \tag{1.30}$$

A mathematician in love may describe the perfectness of their partner by saying that the partner's qualities and bad qualities are disjoint sets. The partner would definitely accept this as a complement.

1.8 Complement (difference)

Complement of set A is a set of elements not in A. In other words, it's the opposite of A.

$$A^C = \{x : x \notin A\} \tag{1.31}$$

The notion of complement (or difference) requires an implicit assumption: what other elements are we talking about? Say, the complement of all positive numbers is negative numbers, and zero, and dogs, and Korean words, and... all possible elements that aren't positive numbers? Thinking this way quickly reduces the conversation to either absurdity or to Russell's paradox. This is why the complement assumes some larger set in which A exists. This larger set should be obvious from the context, or should be specified explicitly somewhere. For instance, when talking about movies, the complement of the set of my favorite movies is obviously all other movies that aren't my favorite.

The term **universe** is used for a collection of entities one wishes to consider. For the example of movies, the universe is probably "all movies ever produced".

It's sometimes useful to consider the complement of a set with respect to some other well-defined set. This is often called **set difference** and is denoted as $A - B$. Formally:

$$A - B = \{x | x \in A \text{ but } x \notin B\} \tag{1.32}$$

SQL operator `EXCEPT` is similar to this idea.

```
SELECT username FROM users
EXCEPT
SELECT username FROM managers;
```

This query would return all users who aren't managers. In set theory, we'd write it this way:

$$\text{Users} - \text{Managers} \tag{1.33}$$

Or, with implicit universe:

$$\text{Managers}^C \tag{1.34}$$

with an assumption that we're talking about users in general.

1.9 Relational algebra

If you worked with SQL in different databases, you might've noticed slight differences or even large, annoying discrepancies. The reason is that SQL isn't a formal, strict language, but rather an approach based on a general formal language called Relational algebra. Developers of different databases design their own versions of SQL and implement and modify its features at their discretion.

Relational algebra was created by Edgar F. Codd in 1960s-1970s, who worked at IBM at the time. Codd proposed this formal language as a basis for database querying.

The good news is that knowing set theory and relational algebra reduces potential SQL-related problems to basically documentation lookup. If you know how unions work, what's the idea behind a Cartesian product and what are expressions of relational algebra, then it comes down to finding the correct SQL syntax to solve the problem at hand.

We will finish up the section on set theory with a short overview of different aspects of relational algebra and their relation to SQL.

1. Relation

   Relation is essentially a table, with columns (called "attributes") and rows. For example, a relation `User` may be used to describe users in a web service.

   ```
   User(id, username, firstName, lastName, email)
   ```

id	username	firstName	lastName	email
1	jenn	Jenn	Clarkson	jnc@hotmail.com
2	pax	Paxi	Romanov	paxro@aol.com
91	thankso	Barack	Liu	bl@wh.gov

   For demonstrating purposes, let's define another table:

   ```
   Manager(id, department, level)
   ```

id	department	level
1	engineering	3
2	medicine	2
91	mathematics	3

   The name of a relation can play a role of the simplest query:

   ```
   User
   ```

   which returns the whole table.

2. Projection

   A **projection** is an operation of extracting records with specific columns (attributes). It is denoted by Greek letter Π (uppercase pi), with a list of columns as subscript, followed by relation's name.

   For example, here we extract a sub-relation of User with ids and emails only:

$$\Pi_{\text{id, email}} \text{ User} \tag{1.35}$$

id	email
1	jnc@hotmail.com
2	paxro@aol.com
91	bl@wh.gov

3. Select

   A **selection** is an operation of filtering records by values. It is denoted by Greek letter σ (sigma) with a condition, followed by relation's name.

   For example, here we get managers with level higher than 2:

$$\sigma_{\text{level}>2} \text{ Manager} \tag{1.36}$$

id	department	level
1	engineering	3
2	medicine	2
91	mathematics	3

4. Operating on expressions

   Select and project operators work on any expression, not only on whole relations. This means we can combine them arbitrarily. For example, we can select (essentially, filter) first, and then extract certain columns.

$$\Pi_{\text{id, level}}(\sigma_{\text{level}>2} \text{ Manager}) \tag{1.37}$$

   The result is a newly constructed relation consisting of only two columns, which, in turn, can be used for other operations:

id	level
1	3
91	3

5. Cross product (cartesian product)

   Recall the idea behind Cartesian product in set theory: given two sets $A = \{a, b\}$ and $B = \{1, 2\}$, the **Cartesian product** is a set of pairs of all combinations of elements:

   $$(a, 1) \qquad (a, 2) \qquad (b, 1) \qquad (b, 2)$$

   (1.38)

   Much of the power of relational algebra comes from applying this idea to relations. It is also often called **cross product**.

   Here's the cross-product of `User` and `Manager`:

   $$\text{User} \times \text{Manager} \tag{1.39}$$

User.id	username	firstName	lastName	email	Manager.id	department
1	jenn	Jenn	Clarkson	jnc@hotmail.com	1	engineering
1	jenn	Jenn	Clarkson	jnc@hotmail.com	2	medicine
1	jenn	Jenn	Clarkson	jnc@hotmail.com	91	mathematics
2	pax	Paxi	Romanov	paxro@aol.com	1	engineering
2	pax	Paxi	Romanov	paxro@aol.com	2	medicine
2	pax	Paxi	Romanov	paxro@aol.com	91	mathematics
91	thankso	Barack	Liu	bl@wh.gov	1	engineering
91	thankso	Barack	Liu	bl@wh.gov	2	medicine
91	thankso	Barack	Liu	bl@wh.gov	91	mathematics

   Since both relations contain a column named `id`, the resulting relation contains two distinct columns with tagged names `User.id` and `Manager.id`. This is similar to the way we "tagged" items in set union.

   This might not seem too useful: we just combined all records, and many of the new rows don't make sense. Cross-product is rarely useful as is. Instead, the goal is often to generate raw data for the operations. For example, with this long table at hand, we can first eliminate the nonsense rows by applying select:

   $$\sigma_{\text{User.id}=\text{Manager.id}}(\text{User} \times \text{Manager}) \tag{1.40}$$

User.id	username	firstName	lastName	email	Manager.id	departr
1	jenn	Jenn	Clarkson	jnc@hotmail.com	1	enginee
2	pax	Paxi	Romanov	paxro@aol.com	2	medicin
91	thankso	Barack	Liu	bl@wh.gov	91	mathen

Assuming user ids in the system are used to identify managers as well, we now have a sensible relation of managers with their complete user info intact.

Now we can filter managers by level and get rid of unneeded columns:

$$\Pi_{\text{User.id, email, department}}\big(\sigma_{\text{User.id=Manager.id,level>2}}(\text{User} \times \text{Manager})\big) \tag{1.41}$$

User.id	email	department
1	jnc@hotmail.com	engineering
91	bl@wh.gov	mathematics

6. Natural join

   The process of performing a cross-product and then filtering out rows that "make sense" by comparing the attributes of the same name is common enough so that relational algebra has a special operator called **natural join**. It is denoted by ⋈ (bow tie). The result is a relation with matching rows and no "tagged" attributes:

$$\text{User} \bowtie \text{Manager} \tag{1.42}$$

id	username	firstName	lastName	email	department	level
1	jenn	Jenn	Clarkson	jnc@hotmail.com	engineering	3
2	pax	Paxi	Romanov	paxro@aol.com	medicine	2
91	thankso	Barack	Liu	bl@wh.gov	mathematics	3

   As you see, natural join is basically a syntactic sugar on top of existing features of relational algebra.

7. Union, intersection, difference

   Relational algebra includes three operators straight from set theory: union, intersection and difference. They work just like you might expect, although there are some caveats. We will not focus on these topics at the moment.

1.10 Relations and Functions

In programming, we often deal with paired data:

- username — email
- person — phone number
- id — company

All cases can be considered as two sets, with a rule to connect their elements.

Given two sets A and B, a *map function* f from A to B, denoted as

$$f : A \to B \tag{1.43}$$

is an assignment to each element a in A of a single element in B.

We sometimes use a notation $a \mapsto f(a)$ (note the different arrow) to indicate that a maps to some value via function f. We call $f(a)$ an *image* of a. Set A is called the *domain* of f, and B the *codomain* of f.

Notice how we talk about functions as static data, not as a process. In programming, we're used to functions being descriptions of processes, even though in most languages we can pass functions around as data. But consider this: how would you cache a function call? An obvious approach is to pre-compute some answers beforehand and store them in a table:

Argument	Return
"a"	52321
"b"	12321
"c	81872
...	...

Theoretically, we could do this for all possible arguments, which would allow us to disregard all function's code and continue operating with a cache table exclusively. This means that any function (unless it deals with randomization) can be represented as static data. A mathematical notation of a function as a relation between two sets (essentially, a set of arguments and a set of return values) generalizes this idea.

A **binary relation** of two sets A and B is a subset of $A \times B$. (Recall that $A \times B$ is a set of all combinations of pairs of values of the two sets). Thus, a *function* is a binary relation, having the property that for each element $a \in A$ there is exactly one ordered pair in $A \times B$ whose first component is a.

There are three types of relations:

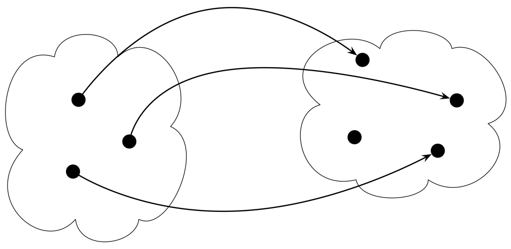


Figure 1.3: Injection.

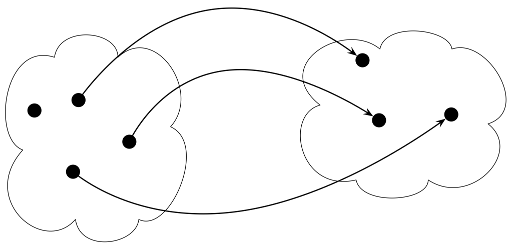


Figure 1.4: Surjection.

1. The function $f : A \to B$ is *one-to-one* (**injective**) if no two values of A result in the same value of B. In other words, there's at most one incoming arrow for each element of B.

Formally: for any two distinct elements a and a' in A, we have $f(a) \neq f(a')$.

2. The function $f : A \to B$ is *onto* (**surjective**) if all elements of B are covered. In other words, no element of B is without at least one arrow.

Formally: for each element $b \in B$, there exists an element $a \in A$, such that $f(a) = b$.

3. The function f is a **bijection** if f is both injective and surjective. In other words, all elements of A are mapped uniquely to all elements of B.

We'll not focus on functions and relations anymore at the moment. Depending on how this course goes on, we'll either keep getting back to elements

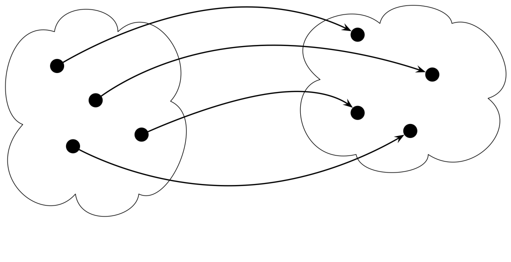


Figure 1.5: Bijection.

of this topic, or will extract it into a separate chapter.

Chapter 2

Proof techniques

2.1 Direct proof

Back when we were discussing set cardinality, we've successfully proved a statement: *If A is a finite set of m elements, then there are 2^m subsets of A.* Let's talk about proofs in more detail, and discuss different proof techniques.

A proof is a sequence of mathematical statements, a path from some basic truth to the desired outcome. An impeccable argument, if you will. Possible the simplest form of proof is a direct proof. It's a direct attempt to see what the statement means if we dare to play with it. Consider a theorem:

Theorem 1. If n is an odd positive integer, then n^2 is odd.

Proof. An odd positive integer can be written as $n = 2k+1$, since something like $2k$ is even and adding 1 makes it definitely odd. We're interested in what odd squared looks like, so let's square this definition:

$$n^2 = (2k+1)^2 =$$

$$4k^2 + 4k + 1 =$$

$$2(2k^2 + 2k) + 1$$

So, we have this final formula $2(2k^2 + 2k) + 1$ and it follows the pattern of $2k + 1$. This means it's odd! We have a proof.

This theorem is based on an idea that numbers described as $2k + 1$ are definitely odd. This might be another theorem that requires another proof, and that proof might be based on some other theorems. The general idea of mathematics is that if you follow any theorem to the very beginning, you'll meet the fundamental axioms, the basis of everything.

2.2 Constructive proof

The idea behind constructive proof is to prove the existence of a certain object by describing a method of creating it.

Theorem 2. There exists a rational number x such that $\sqrt{10^{100}} < x < \sqrt{10^{100}+1}$.

Proof. $\sqrt{10^{100}}$ is 10^{50}. Now, let's just try different clearly rational values for x that *seem* to lie between the required boundaries. For example, let's try $x = 10^{50} + 10^{51}$. It's obviously larger than 10^{50}, so we only need to show that it's smaller than $\sqrt{10^{100}+1}$.

To do that, let's compute x^2 so that we can compare it with $(\sqrt{10^{100}+1})^2$:

$$x^2 = ((10^{50} + 10^{-51})^2$$
$$= 10^{100} + 2 \times (10^{50} \times 10^{-51}) + 10^{-102}$$
$$= 10^{100} + 2 \times 10^{50-51} + 10^{-102}$$
$$= 10^{100} + 2 \times 10^{-1} + 10^{-102}$$

Clearly, $2 \times 10^{-1} + 10^{-102}$ is less than 1, so $x^2 < (\sqrt{10^{100}+1})^2$, and, by extension, $x < \sqrt{10^{100}+1}$.

2.3 Proof by Contradiction

Another approach is proof by contradiction. Here is the idea:

1. Assume the statement is false or true.
2. Derive a contradiction, a paradox, something that doesn't make sense. This will mean that the statement cannot possibly be what we assumed it to be, therefore it's the opposite.

When I first saw this formal technique, it puzzled me. It didn't seem to be valid: alright, assuming something is false leads to a paradox, so what? We haven't proven that assuming it's true doesn't lead to another paradox! Or even the same paradox, for that matter. What I failed to understand conceptually is that a statement is a binary thing: it's either true or untrue. Nothing in between. So, if one can definitely declare "X is not false", then no other options are left: "X must be true".

Theorem 3. n is a positive integer. If n^2 is even, then n is even.

We may try to construct another direct proof, but creating paradoxes is much more fun!

Proof. Let's assume that n^2 is even, **but n is odd**. This is the opposite of what we want, and we will show that this scenario is impossible.

n is odd, then, according to Theorem 3, n^2 must be odd. This doesn't make sense! Our assumption and our conclusion are the opposite. This is a paradox, so the assumption was wrong. Meaning, the idea "n^2 is even, but n is odd" is false. Therefore, the idea "n^2 is even, n is even" is true.

Theorem 4. $\sqrt{2}$ is irrational.

Woah, this is... different. Prior theorems were formulas, something to play with, something physical. But here is just an idea, so how would we even start?

Let's start with a definition.

> In mathematics, the irrational numbers are all real numbers which are not rational numbers.[1]

Doesn't seem helpful, but let's continue. What are rational numbers then? Are they some reasonable beings who make optimal decisions all the time?

> A rational number is any number that can be expressed as the fraction $\frac{p}{q}$ of two integers.[2]

Oh! They are rational because they are *ratios*! Just to make things super clear, let's dig one more step and make sure we understand integers.

> An integer (from the Latin *integer* meaning "whole") is a number that can be written without a fractional component. For example, 21, 4, 0, and 2048 are integers, while 9.75, $5\frac{1}{2}$ and $\sqrt{2}$ are not.[3]

Combining these things, we can construct a comprehensive definition of the irrational number: it's a number that cannot be expressed as the fraction of two whole numbers. Now, let's apply this to Theorem 3 so that it has some meat:

Theorem 4, re-framed. $\sqrt{2}$ cannot be expressed as $\frac{p}{q}$, where p and q are integers.

Alright, now there is something to play with!

Proof. Start by assuming the opposite – $\sqrt{2}$ is rational. This means it can be written as a fraction of two integers:

[1] `https://en.wikipedia.org/wiki/Irrational_number`
[2] `https://en.wikipedia.org/wiki/Rational_number`
[3] `https://en.wikipedia.org/wiki/Integer`

$$\sqrt{2} = \frac{p}{q}$$

We can assume that p and q are not **both** even, because if they are, we can reduce them by dividing both by a common factor (like, for example, $\frac{8}{10}$
$)should be reduced to \frac{4}{5}$
$)). In other words, if they are both even, reduce them until at least one is odd and no further reductic$

Now, let's square the square root:

$$(\sqrt{2})^2 = \frac{p^2}{q^2}$$

$$2 = \frac{p^2}{q^2}$$

$$p^2 = 2q^2$$

Remember, something like $2k + 1$ is odd, and $2k$ is even. Here we see this pattern: $p^2 = 2q^2$, which means that p^2 is even (it consists of *two* things).

Then, using Theorem 3, we can say that p is even as well, which means we can write p as $p = 2k$. So:

$$2q^2 = p^2 = (2k)^2$$

$$2q^2 = 4k^2$$

Divide both by two:

$$q^2 = 2k^2$$

So, q^2 is even. By the same Theorem 2 it follows that q is even.

Let's summarize the two conclusions:

1. p is even.
2. q is even.

Wait... We made sure that not both p and q are even before starting this whole thing! We made sure to reduce them until at least one is odd, but then, by applying Theorem 2, we ended up with two even numbers. This is impossible, so the idea that "$\sqrt{2}$ is rational" is not true.

Therefore, $\sqrt{2}$ is irrational.

2.4 Proof by Induction

One of the most powerful techniques is proof by induction.

Do not confuse mathematical induction with inductive or deductive reasoning. Despite the name, mathematical induction is actually a form of deductive reasoning.

Let's say, we want to prove that some statement P is true for all positive integers. In other words:

$P(1)$ is true, $P(2)$ is true, $P(3)$ is true… etc.

We could try and prove each one directly or by contradiction, but the infinite number of positive integers makes this task rather grueling. Proof by induction is a sort of generalization that starts with the basis:

Basis: Prove that $P(1)$ is true.

Then makes one generic step that can be applied indefinitely:

Induction step: Prove that for all $n \geq 1$, the following statement holds: If $P(n)$ is true, then $P(n+1)$ is also true.

We've devised another problem to solve, and it's seemingly the same. But if the basis is true, then proving this *inductive step* will prove the theorem. To do this, we chose an arbitrary $n \geq 1$ and assumed that $P(n)$ is true. This assumption is called the *inductive hypothesis*. The tricky part is this: we don't prove the hypothesis directly, but prove the $n+1$ version of it.

This is all rather amorphous, so let's prove a real theorem.

Theorem 5. For all positive integers n, the following is true:

$$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2} \tag{2.1}$$

Proof. Start with the basis when n is 1. Just calculate it:

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

This is correct, so, the basis is proven. Now, assume that the theorem is true for any $n \geq 1$:

$$1 + 2 + 3 + ... + n = \frac{n(n+1)}{2} \tag{2.2}$$

In the induction step we have to prove that it's true for $n+1$:

$$1 + 2 + 3 + ... + (n+1) = \frac{(n+1)(n+2)}{2} \tag{2.3}$$

Having this equation, we should just try to expand it and prove directly. Since the last member on the left side is $n+1$, the second last must be n, so:

$$1 + 2 + 3 + ... + (n + 1) = 1 + 2 + 3 + ... + n + (n + 1)$$

From our assumption, we know, that

$$1 + 2 + 3 + ... + n = \frac{n(n + 1)}{2}.$$

So, let's replace it on the right hand side:

$$1 + 2 + 3 + ... + (n + 1) = \frac{n(n + 1)}{2} + (n + 1)$$

And then make that addition so that the right hand side is a single fraction:

$$\begin{aligned} 1 + 2 + 3 + ... + (n + 1) &= \frac{n(n + 1)}{2} + \frac{2(n + 1)}{2} \\ &= \frac{n(n + 1) + 2(n + 1)}{2} \\ &= \frac{(n + 1)(n + 2)}{2}. \end{aligned}$$

Done, we have proven that the inductive step (2.3) is true.

There are two results:

1. The theorem is true for $n = 1$.
2. If the theorem is true for any n, then it's also true for $n + 1$.

Combining these two results we can conclude that the theorem is true for all positive integers n.

I had troubles with this technique because for a long time I couldn't for the life of me understand why is this *enough* and how is the basis *helping*?! The basis seemed redundant. We assume $P(n)$ is true, then prove that $P(n + 1)$ is true given that $P(n)$ is true, but so what? We didn't prove the thing we assumed!

It clicked after I understood that we don't have to prove $P(n)$, we just take the concrete value from the basis and use it as n. Since we have a proof of $P(n + 1)$ being true **if** $P(n)$ is true, we conclude that if $P(1)$ is true, then $P(1 + 1)$ is true.

Well, if $P(1 + 1)$ is true, then, using the same idea, $P(1 + 1 + 1)$ is true, and so forth.

The basis was the cheat-code to kick-start the process by avoiding the need to prove the assumption 2.2.

2.5 The pigeon hole principle