Lectures Notes on Selected Mathematical Preliminaries of Computer Science

Florian Rabe

2018

Contents

Chapter 1

Mathematical Preliminaries

1.1 Power Sets

1.1.1 Subsets and Characteristic Functions

1.1.2 Operations on Subsets

1.1.3 Closures

Definition 1.1 (Closure System). A **closure system** C on a set A is a set of subsets of A (called the **closed** sets) such that the intersection of any collection of closed sets is again closed.

For any $X \subseteq A$, the smallest closed superset of X (which can be constructed by taking the intersection of all closed supersets of X) is called the C-**closure** of X.

Here "any number" is meant to include the intersection of

- infinitely many sets
- no set, i.e., A (which is the intersection of 0 subsets of A) is always closed

Definition 1.2 (Closure Operator). A **closure operator** C on a set A is a function from subsets of A to subsets of A such that

- for all $X \subseteq A$: $X \subseteq C(X)$
- for all $X \subseteq A$: $C(C(X)) = C(X)$
- for all $X, Y \subseteq A$: if $X \subseteq Y$, then $C(X) \subseteq C(Y)$

The sets for which $C(X) = X$ are called C-**closed**.

Theorem 1.3 (Closure). *Closure systems and closure operators are the same in the following sense:*

- *Given a closure operator, the closed sets form a closure system.*
- *Given a closure system, the closure is a closure operator.*

Closure operators occur in multiple places of these notes including

Example 1.4. Given a set U, the reflexive/symmetric/transitive/reflexive-transitive/equivalence relations on U each form a closure system on $U \times U$. The respective closure operators are

- reflexive closure: $r \mapsto r \cup \Delta_U$
- symmetric closure: $r \mapsto r \cup r^{-1}$
- transitive closure: $r \mapsto r \cup rr \cup rrr \cup \ldots = \bigcup_{i=1} r^i$
- reflexive-transitive closure: $r \mapsto \Delta_U \cup r \cup rr \cup rrr \cup \ldots = \bigcup_{i=0} r^i$
- equivalence closure: $r \mapsto \bigcup_{i=0}(r \cup r^{-1})^i$

1.2 Relations and Functions

A binary relation between sets A and B is a subset $\# \subseteq A \times B$. We usually write $(x, y) \in \#$ as $x\#y$.

1.2.1 Classification

Definition 1.5 (Properties of Relations)**.** We say that # is ... if the following holds:

- functional: for all $x \in A$, there is at most one $y \in B$ such that $x\#y$.
- total: for all $x \in A$, there is at least one $y \in B$ such that $x\#y$.
- injective: for all $y \in B$, there is at most one $x \in A$ such that $x\#y$.
- surjective: for all $y \in B$, there is at least one $x \in A$ such that $x\#y$.

Moreover, # is ... if it is ... :

- a partial function: functional
- a (total) function: functional and total
- bijective: total, functional, injective, and bijective

If a # is functional, we write $\#(x)$ for the $y \in B$ such that $x\#y$ if such a (necessarily unique) y exists.

Because relations are the subsets of some set (namely $A \times B$), they admit all operations of subsets:

- empty relation: for no $x \in A$, $y \in B$ we have $x\#y$
- universal relation: for all $x \in A$, $y \in B$ we have $x\#y$
- for two relations r and s:
  - union of r and s: $x\,(r \cup s)\,y$ iff $x\,r\,y$ or $x\,r\,y$
  - intersection of r and s: $x\,(r \cup s)\,y$ iff $x\,r\,y$ and $x\,r\,y$

Moreover, if $r \subseteq s$ (if $x\,r\,y$, then also $x\,s\,y$), we say that s is a refinement of r.

1.2.2 Operations on Relations

Definition 1.6 (Identity)**.** We define the identity relation Δ_A between A and A by:

$$x\,\Delta_A\,y \qquad \text{iff} \qquad x = y$$

i.e., $\Delta_A = \{(x, x) : x \in A\}$.

Theorem 1.7 (Identity)**.** *The identity is functional, total, injective, and surjective.*

Definition 1.8 (Composition)**.** Given relations r between A and B and s between B and C, we define the relation rs between A and C by

$$x\,(rs)\,z \qquad \text{iff} \qquad x\,r\,y \text{ and } y\,s\,z \text{ for some } y \in B$$

Composition rs is also written as $r; s$ or $s \circ r$.

Theorem 1.9 (Composition)**.** *If two relations are both functional/total/injective/surjective, then so is their composition.*

Theorem 1.10 (Properties of Composition)**.** *Composition is associative whenever defined.*
Identity is a neutral element for composition whenever defined.

Definition 1.11 (Powers)**.** Given a relation r between A and A and a natural number n, we define X^n inductively by

$$r^0 = \Delta_A$$

$$r^{n+1} = X^n r$$

We also define

$$r^{-n} = (r^{-1})^n$$

Definition 1.12 (Dual Relation)**.** For every relation $\#$, the relation $\#^{-1}$ is defined by $x \#^{-1} y$ iff $y \# x$. $\#^{-1}$ is called the **dual** of $\#$.

Theorem 1.13 (Dual Relation)**.** *If a relation is functional/total/injective/surjective, then its dual is injective/surjective/functional/total, respectively.*
In particular, the dual of a bijective function is a bijective function, which we call its ***inverse****.*

Theorem 1.14 (Properties of Dualization)**.** *For all relations r, we have $(r^{-1})^{-1} = r$.*
For the identity relation, we have $\Delta_A^{-1} = \Delta_A$.
For all composed relations, we have $(rs)^{-1} = s^{-1}r^{-1}$.
For all powers of relations, we have $(r^n)^{-1} = (r^{-1})^n = r^{-n}$.

1.2.3 Point-Free Formulations

A statement about relations between A and B is called *point-free* if no elements $x \in A$ or $y \in B$ are mentioned explicitly. Point-free formulations take some getting used to but are more compact and eventually easier to work with.

Theorem 1.15 (Point-Free Formulations)**.** *For a relation r between A and B, the following are equivalent:*

functional	$r^{-1}r \subseteq \Delta_B$
total	$rr^{-1} \supseteq \Delta_A$
injective	$rr^{-1} \subseteq \Delta_A$
surjective	$r^{-1}r \supseteq \Delta_B$

For a relation r on A, the following[1] *are equivalent:*

reflexive	$\Delta_A \subseteq r$
symmetric	$r^{-1} = r$
transitive	$rr \subseteq r$
reflexive and transitive	$\Delta_A \subseteq r$ *and* $rr = r$

Moreover, we have the following equalities for a function f:

$$\ker f = f\, f^{-1} \qquad \operatorname{im} f \times \operatorname{im} f = f^{-1} f$$

1.3 Binary Relations on a Set

In this section, we consider a binary relation $\#$ on a set A, i.e., a subset $\# \subseteq A \times A$.

1.3.1 Classification

[1] The concepts on the left are defined in Def. 1.16.

Definition 1.16 (Properties of Binary Relations)**.** We say that # is ... if the following holds:

- reflexive: for all x, $x\#x$
- irreflexive[2]: for no x, $x\#x$
- symmetric: for all x, y, if $x\#y$, then $y\#x$
- anti-symmetric[3]: for all x, y, if $x\#y$ and $y\#x$, then $x = y$ (= if $x\#y$ then not $y\#x$ (unless $x = y$))
- transitive: for all x, y, z, if $x\#y$ and $y\#z$, then $x\#z$

Moreover, we call # a ... if it is:

- strict order: irreflexive and transitive
- preorder: reflexive and transitive
- order[4]: preorder and anti-symmetric (= reflexive, transitive, and anti-symmetric)
- total[5]order: order and for all x, y, $x\#y$ or $y\#x$
- partial equivalence: symmetric and transitive (= not necessarily reflexive equivalence)
- equivalence: preorder and symmetric (= reflexive, transitive, and symmetric)

An element $a \in A$ is called ... of # if the following holds:

- least element: for all x, $a\#x$
- greatest element: for all x, $x\#a$
- least upper bound of x, y: $x\#a$ and $y\#a$ and for all z, if $x\#z$ and $y\#z$, then $a\#z$
- greatest lower bound of x, y: $a\#x$ and $a\#y$ and for all z, if $z\#x$ and $z\#y$, then $z\#a$

Theorem 1.17 (Dual Relation)**.** *If a relation is reflexive/irreflexive/symmetric/antisymmetric/transitive/total, then so is its dual.*

If a is a least/greatest element for a relation, then it is a greatest/least element for its dual. If a is a least upper/greatest lower bound of x, y for some relation, then it is a greatest lower/least upper bound of x, y for the dual.

1.3.2 Equivalence Relations

Symbols Equivalence relations are usually written using infix symbols whose shape is reminiscent of horizontal lines, such as $=$, $\sim$, or $\equiv$. Often vertically symmetric symbols are used to emphasize the symmetry property.

Equivalence Classes and Quotients Equivalence relations allow grouping related elements into classes and collecting all the classes in what is called a quotient.

Definition 1.18 (Quotient)**.** Consider a relation $\equiv$ on A. Then

- For $x \in A$, the set $\{y \in A \mid x \equiv y\}$ is called the (equivalence) **class** of x. It is often written as $[x]_\equiv$.
- $A/\equiv$ is the set of all classes. It is called the **quotient** of A by $\equiv$.

Definition 1.19 (Kernel)**.** Consider a function $f : A \to B$. The **kernel** of f, written $\ker f$, is the binary relation on A defined by $x\,(\ker f)\,y$ iff $f(x) = f(y)$.

Definition 1.20 (Partition)**.** A **partition** P on a set A is a set of non-empty, pairwise disjoint subsets of A whose overall union is A.

Theorem 1.21. *For a relation $\equiv$ on A, the following are equivalent*[6]*:*

- *$\equiv$ is an equivalence.*
- *$\equiv$ is the kernel of some function, i.e., there is a set B and a function $f : A \to B$ such that $x \equiv y$ iff $f(x) = f(y)$.*
- *$A/\equiv$ is a partition on A, i.e., every element of A is in exactly one class in $A/\equiv$.*

[2]That's not the same as not being reflexive.
[3]That's not the same as not being symmetric.
[4]Orders are also called *partial order*, *poset* (for partially ordered set), or *ordering*.
[5]This notion of *total* has nothing to do with the one from Def. 1.5 of the same name.
[6]Logical equivalence is itself an equivalence relation.

Partial Equivalence Relations Consider a partial equivalence relation $\equiv$ on A. $\equiv$ is not an equivalence because it is not reflexive. However, we can easily prove: if $x \equiv y$, then $x \equiv x$ and $y \equiv y$. Thus, the only elements for which $x \equiv x$ does not hold are the ones that are in relation to no element at all.

Thus, we have:

Theorem 1.22. *A partial equivalence relation $\equiv$ on A is the same as an equivalence relation on a subset of A.*

Normal and Canonical Forms Instead of working with equivalence classes, we usually prefer working with representatives, i.e., designated elements of the classes that we use instead of the entire class.

Definition 1.23 (System of Representatives)**.** Consider an equivalence relation $\equiv$ on A. A subset R of A is a system of **representatives** for $\equiv$ if it contains exactly one element from every $\equiv$-class.

Normal forms are used to choose representatives for each element:

Definition 1.24 (Normal Forms)**.** Consider an equivalence relation $\equiv$ on A. A function $N : A \to A$ is called a ... if

- **normal form**: $N(x) \equiv x$ and $N(N(x)) = N(x)$ for all $x \in A$
- **canonical form**: N is a normal form and $N(x) = N(y)$ whenever $x \equiv y$

We also call $N(x)$ the normal/canonical form of x. The process of mapping x to $N(x)$ is called **normalization**.

The main application of canonical forms is that we can check $x \equiv y$ by comparing $N(x)$ and $N(y)$.

Theorem 1.25. *Consider an equivalence relation $\equiv$ on A and a normal form $N : A \to A$. The following are equivalent:*

- *N is a canonical form.*
- *The image of N is a system of representatives.*
- *$\equiv$ is the kernel of N.*

1.3.3 Orders

Theorem 1.26 (Strict Order vs. Order)**.** *For every strict order $<$ on A, the relation "$x < y$ or $x = y$" is an order.*

For every order $\leq$ on A, the relation "$x \leq y$ and $x \neq y$" is a strict order.

Thus, strict orders and orders come in pairs that carry the same information.

Strict orders are usually written using infix symbols whose shape is reminiscent of a semi-circle that is open to the right, such as $<$, $\subset$, or $\prec$. This emphasizes the anti-symmetry ($x < y$ is very different from $y < x$.) and the transitivity ($< \ldots <$ is still $<$.) The corresponding order is written with an additional horizontal bar at the bottom, i.e., $\leq$, $\subseteq$, or $\preceq$. In both cases, the mirrored symbol is used for the dual relation, i.e., $>$, $\supset$, or $\succ$, and $\geq$, $\supseteq$, and $\succeq$.

Theorem 1.27. *If $\leq$ is an order, then least element, greatest element, least upper bound of x, y, and greatest lower bound of x, y are unique whenever they exist.*

Theorem 1.28 (Preorder vs. Order)**.** *For every preorder $\leq$ on A, the relation "$x \leq y$ and $y \leq x$" is an equivalence.*

For equivalence classes X and Y of the resulting quotient, $x \leq y$ holds for either all pairs or no pairs $(x, y) \in X \times Y$. If it holds for all pairs, we write $X \leq Y$.

The relation $\leq$ on the quotient is an order.

Remark 1.29 (Order vs. Total Order). If $\leq$ is a preorder, then for all elements x, y, there are four mutually exclusive options:

	$x \leq y$	$x \geq y$	$x = y$
x strictly smaller than y, i.e., $x < y$	true	false	false
x strictly greater than y, i.e., $x > y$	false	true	false
x and y incomparable	false	false	false
x and y similar	true	true	maybe

Now anti-symmetry excludes the option of similarity (except when $x = y$ in which case trivially $x \leq y$ and $x \geq y$). And totality excludes the option of incomparability.

Combining the two exclusions, a total order only allows for $x > y$, $y < x$, and $x = y$.

1.4 Binary Functions on a Set

A binary function on A is a function $\circ : A \times A \to A$. We usually write $\circ(x, y)$ as $x \circ y$.

Definition 1.30 (Properties of Binary Functions)**.** We say that $\circ$ is …if the following holds:

- associative: for all x, y, z, $x \circ (y \circ z) = (x \circ y) \circ z$
- commutative: for all x, y, $x \circ y = y \circ x$
- idempotent: for all x, $x \circ x = x$

An element $a \in A$ is called a …element of $\circ$ if the following holds:

- left-neutral: for all x, $a \circ x = x$
- right-neutral: for all x, and $x \circ a = x$
- neutral: left-neutral and right-neutral
- left-absorbing: for all x, $a \circ x = a$
- right-absorbing: for all x, $x \circ a = a$
- absorbing: left-absorbing and right-absorbing
- if e is a neutral element:
  - left-inverse of x: $a \circ x = e$
  - right-inverse of x: $x \circ a = e$
  - inverse of x: left-inverse and right-inverse of x

Moreover, we say that $\circ$ is a …if it is/has:

- semigroup: associative
- monoid: associative and neutral element
- group: monoid and inverse elements for all x
- semilattice: associative, commutative, and idempotent
- bounded semilattice: semilattice and neutral element

Terminology 1.31. The terminology for *absorbing* is not well-standardized. *Attractive* is an alternative word sometimes used instead.

Theorem 1.32. *Neutral and absorbing element of $\circ$ are unique whenever they exist.*
If $\circ$ is a monoid, then the inverse of x is unique whenever it exists.

1.5 The Integer Numbers

1.5.1 Divisibility

Definition 1.33 (Divisibility)**.** For $x, y \in \mathbb{Z}$, we write $x|y$ iff there is a $k \in \mathbb{Z}$ such that $x * k = y$.
We say that y is divisible by x or that x divides y.

Remark 1.34 (Divisible by 0 and 1). Even though division by 0 is forbidden, the case $x = 0$ is perfectly fine. But it is boring: $0|x$ iff $x = 0$.
Similarly, the case $x = 1$ is trivial: $1|x$ for all x.

Theorem 1.35 (Divisibility). *Divisibility has the following properties for all $x, y, z \in Z$*

- *reflexive:* $x|x$
- *transitive: if $x|y$ and $y|z$ then $x|z$*
- *anti-symmetric for natural numbers $x, y \in \mathbb{N}$: if $x|y$ and $y|x$, then $x = y$*
- 1 *is a least element:* $1|x$
- 0 *is a greatest element:* $x|0$
- $\gcd(x, y)$ *is a greatest lower bound of* x, y
- $\mathrm{lcm}(x, y)$ *is a least upper bound of* x, y

Thus, $|$ *is a preorder on* $\mathbb{Z}$ *and an order on* $\mathbb{N}$.

Divisibility is preserved by arithmetic operations: If $x|m$ and $y|m$, then

- *preserved by addition:* $x + y|m$
- *preserved by subtraction:* $x - y|m$
- *preserved by multiplication:* $x * y|m$
- *preserved by division if $x/y \in \mathbb{Z}$:* $x/y|m$
- *preserved by negation of any argument:* $-x|m$ *and* $x| - m$

gcd *has the following properties for all $x, y \in \mathbb{N}$:*

- *associative:* $\gcd(\gcd(x, y), z) = \gcd(x, \gcd(y, z))$
- *commutative:* $\gcd(x, y) = \gcd(y, x)$
- *idempotent:* $\gcd(x, x) = x$
- 0 *is a neutral element:* $\gcd(0, x) = x$
- 1 *is an absorbing element:* $\gcd(1, x) = 1$

lcm *has the same properties as* gcd *except that* 1 *is neutral and* 0 *is absorbing.*

Theorem 1.36. *For all $x, y \in \mathbb{Z}$, there are numbers $a, b \in \mathbb{Z}$ such that* $ax + by = \gcd(x, y)$.
a and b can be computed using the extended Euclidean algorithm.

Definition 1.37. If $\gcd(x, y) = 1$, we call x and y **coprime**.
For $x \in \mathbb{N}$, the number of coprime $y \in \{0, \ldots, x - 1\}$ is called $\varphi(x)$. φ is called Euler's **totient function**.

Example 1.38. We have $\varphi(0) = 0$, $\varphi(1) = \varphi(2) = 1$, $\varphi(3) = \varphi(4) = 2$, and so on. Because $\gcd(x, 0) = x$, we have $\varphi(x) \leq x - 1$. x is prime iff $\varphi(x) = x - 1$.

1.5.2 Equivalence Modulo

Definition 1.39 (Equivalence Modulo). For $x, y, m \in \mathbb{Z}$, we write $x \equiv_m y$ iff $m|x - y$.

Theorem 1.40 (Relationship between Divisibility and Modulo). *The following are equivalent:*

- $m|n$
- $\equiv_m \supseteq \equiv_n$ *(i.e., for all x, y we have that $x \equiv_n y$ implies $x \equiv_m y$)*
- $n \equiv_m 0$

Remark 1.41 (Modulo 0 and 1). In particular, the cases $m = 0$ and $m = 1$ are trivial again:

- $x \equiv_0 y$ iff $x = y$,
- $x \equiv_1 y$ always

Thus, just like 0 and 1 are greatest and least element for $|$, we have that $\equiv_0$ and $\equiv_1$ are the smallest and the largest equivalence relation on $\mathbb{Z}$.

Theorem 1.42 (Modulo). *The relation $\equiv_m$ has the following properties*

- *reflexive: $x \equiv_m x$*
- *transitive: if $x \equiv_m y$ and $y \equiv_m z$ then $x \equiv_m z$*
- *symmetric: if $x \equiv_m y$ then $y \equiv_m x$*

Thus, it is an equivalence relation.

It is also preserved by arithmetic operations: If $x \equiv_m x'$ and $y \equiv_m y'$, then

- *preserved by addition: $x + y \equiv_m x' + y'$*
- *preserved by subtraction: $x - y \equiv_m x' - y'$*
- *preserved by multiplication: $x \cdot y \equiv_m x' \cdot y'$*
- *preserved by division if $x/y \in \mathbb{Z}$ and $x'/y' \in \mathbb{Z}$: $x/y \equiv_m x'/y'$*
- *preserved by negation of both arguments: $-x \equiv_m -x'$*

1.5.3 Arithmetic Modulo

Definition 1.43 (Modulus). We write $x \bmod m$ for the smallest $y \in \mathbb{N}$ such that $x \equiv_m y$.
We also write $modulus_m$ for the function $x \mapsto x \bmod m$. We write $\mathbb{Z}_m$ for the image of $modulus_m$.

Theorem 1.44 (Modulus). *$modulus_m$ and $\mathbb{Z}_m$ are a canonical form and a system of representatives for $\equiv_m$.*

Remark 1.45 (Modulo 0 and 1). The cases $m = 0$ and $m = 1$ are trivial again:

- $x \bmod 0 = x$ and $\mathbb{Z}_0 = \mathbb{Z}$
- $x \bmod 1 = 0$ and $\mathbb{Z}_1 = \{0\}$

Remark 1.46 (Possible Values). For $m \neq 0$, we have $\mathbb{Z}_m = \{0, \ldots, m-1\}$. In particular, there are m possible values for $x \bmod m$.
For example, we have $x \bmod 1 \in \{0\}$. And we have $x \bmod 2 = 0$ if x is even and $x \bmod 2 = 1$ if x is odd.

Definition 1.47 (Arithmetic Modulo m). For $x, y \in \mathbb{Z}$, we define arithmetic operations modulo m by

$$x \circ_m y = (x \circ y) \bmod m \qquad \text{for} \qquad \circ \in \{+, -, \cdot\}$$

Moreover, if there is a unique $q \in \mathbb{Z}_m$ such that $q \cdot x \equiv_m y$, we define $x/_m y = q$.

Note that the condition $y|x$ is neither necessary nor sufficient for $x/_m y$ to de defined. For example, $2/_4 2$ is undefined because $1 \cdot 2 \equiv_4 3 \cdot 2 \equiv_4 2$. Conversely, $2/_4 3$ is defined, namely 2.

Theorem 1.48 (Arithmetic Modulo m). *For $x, y \in \mathbb{Z}$,* mod *commutes with arithmetic operations in the sense that*

$$(x \circ y) \bmod m = (x \bmod m) \circ_m (y \bmod m) \qquad \text{for} \qquad \circ \in \{+, -, \cdot\}$$

Moreover, $x/_m y$ is defined iff $\gcd(y, m) = 1$ and

$$(x/y) \bmod m = (x \bmod m)/_m (y \bmod m) \qquad \text{if} \qquad y|x$$

$$x/_m y = x \cdot_m a \qquad \text{if } ay + bm = 1 \textit{ as in Thm. 1.36}$$

Theorem 1.49 (Fermat's Little Theorem). *For all prime numbers p and $x \in \mathbb{Z}$, we have that $x^p \equiv_p x$.
If x and p are coprime, that is equivalent to $x^{p-1} \equiv_p 1$.*

1.5.4 Digit-Base Representations

Fix $m \in \mathbb{N} \setminus \{0\}$, which we call the base.

Theorem 1.50 (Div-Mod Representation)**.** *Every $x \in \mathbb{Z}$ can be uniquely represented as $a \cdot m + b$ for $a \in \mathbb{Z}$ and $b \in \mathbb{Z}_m$.*
Moreover, $b = x \bmod m$. We write $b \operatorname{div} m$ for a.

Definition 1.51 (Base-m-Notation)**.** For $d_i \in \mathbb{Z}_m$, we define $(d_k \dots d_0)_m = d_k \cdot m^k + \dots + d_1 \cdot k + d_0$.
The d_i are called **digits**.

Theorem 1.52 (Base-m Representation)**.** *Every $x \in \mathbb{N}$ can be uniquely represented as $(0)_m$ or $(d_k \dots d_0)_m$ such that $d_k \neq 0$.*
Moreover, we have $k = \lfloor \log_m x \rfloor$ and $d_0 = x \bmod m$, $d_1 = (x \operatorname{div} m) \bmod m$, $d_2 = ((x \operatorname{div} m) \operatorname{div} m) \bmod m$ and so on.

Example 1.53 (Important Bases)*.* We call $(d_k \dots d_0)_m$ the binary/octal/decimal/hexadecimal representation if $m = 2, 8, 10, 16$, respectively.
In case $m = 16$, we write the elements of $\mathbb{Z}_m$ as $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f\}$

1.5.5 Finite Fields

In this section, let $m = p$ be prime.

Construction Because p is prime, $x/_p y$ is defined for all $x, y \in \mathbb{Z}_p$ with $y \neq 0$. Moreover, $\mathbb{Z}_p$ is a field.

Up to isomorphism, all finite fields are obtained as n-dimensional vector spaces $\mathbb{Z}_p^n$ for some prime p and $n \geq 1$. This field is usually called F_{p^n} because it has p^n elements. From now on, let $q = p^n$.

The elements of F_q are vectors $(a_0, \dots, a_{n-1})$ for $a_i \in \mathbb{Z}_p$. Addition and subtraction are component-wise, the 0-element is $(0, \dots, 0)$, the 1-element is $(1, 0, \dots, 0)$.

However, multiplication in F_q is tricky if $n > 1$. To multiply two elements, we think of the vectors $(a_0, \dots, a_{n-1})$ as polynomials $a_{n-1}X^{n-1} + \dots + a_1 X + a_0$ and multiply the polynomials. This can introduce powers X^n and higher, which we eliminate using $X^n = k_{n-1}X^{n-1} + \dots + k_1 X + k_0$ for certain k_i. The resulting polynomial has degree at most $n - 1$, and its coefficients (modulo p) yield the result.

The values k_i always exists but are non-trivial to find. They must be such that the polynomial $X^n - k_{n-1}X^{n-1} - \dots - k_1 X - k_0$ has no roots in $\mathbb{Z}_p$. There may be multiple such polynomials, which may lead to different multiplication operations. However, all of them yield isomorphic fields.

Binary Fields The operations become particularly easy if $p = 2$. The elements of F_{2^n} are just the bit vectors of length n. Addition and subtraction are the same operation and can be computed by component-wise XOR. Multiplication is a bit more complex but can be obtained as a sequence of bit-shifts and XORs.

Exponentiation and Logarithm Because F_q has multiplication, we can define natural powers in the usual way:

Definition 1.54. For $x \in F_q$ and $l \in \mathbb{N}$, we define $x^l \in F_q$ by $x^0 = 1$ and $x^{l+1} = x \cdot x^l$.
If $l \in \mathbb{N}$ is the smallest number such that $x^l = y$, we write $l = \log_x y$ and call n the **discrete q-logarithm** of y with base x.

The powers $1, x, x^2, \ldots \in F_q$ of x can take only $q - 1$ different values because F_q has only q elements and x^l can never be 0 (unless $x = 0$). Therefore, they must be periodic:

Theorem 1.55. *For every $x \in F_q$, we have $x^q = x$. If $x \neq 0$, that is equivalent to $x^{q-1} = 1$.*

For some x, the period is indeed $q-1$, i.e., we have $\{1, x, x^2, \ldots, x^{q-1}\} = F_q \setminus \{0\}$. Such an x is called a **primitive element** of F_q. In that case $\log_x y$ is defined for all y.

But the period may be smaller. For example, the powers of 1 are $1, \ldots, 1$, i.e., 1 has period 1. For a non-trivial example consider $p = 5$, $n = 1$, (i.e., $q = 5$): The powers of 4 are $4^0 = 1$, $4^1 = 4$, $4^2 = 16 \bmod 5 = 1$, and $4^3 = 4$.

If the period is smaller than $q-1$, x^l does not take all possible values in F_q. In that case $\log_x y$ is not defined for all $y \in F_q$.

Computing x^l is straightforward and can be done efficiently. (If $n > 1$, we first have to find the values k_i needed to do the multiplication, but we can precompute them once and for all.)

Determining whether $\log_x y$ is defined and computing its value is also straightforward: We can enumerate all powers $1, x, x^2, \ldots$ until $x^l = 1$ (in which case the logarithm is undefined) or $x^l = y$ (in which case the logarithm is l). However, no efficient algorithm is known.

1.5.6 Infinity

Occasionally, it is useful to compute also with infinity ∞ or $-\infty$. When adding infinity, some but not all arithmetic operations still behave nicely.

Positive Infinity We write $\mathbb{N}^\infty = \mathbb{N} \cup \{\infty\}$.

The order $\leq$ works as usual. ∞ is the greatest element.

Addition works as usual. ∞ is an attractive element.

Subtraction is introduced as usual, i.e., $a - b = x$ whenever x is the unique value such that $a = x + b$. Thus, $\infty - n = \infty$ for $n \in \mathbb{N}$. $x - \infty$ is undefined. The law $x - x = 0$ does not hold anymore.

Multiplication becomes partial because $\infty \cdot 0$ is undefined. For $x \neq 0$, we put $\infty \cdot x = \infty$.

Divisibility $|$ is defined as usual. Thus, we have $x|\infty$ for all $x \neq 0$, and $\infty|x$ iff $x = \infty$. There is no greatest element anymore because: 0 and ∞ are both greater than every other element except for each other.

Negative Infinity We write $\mathbb{Z}^\infty = \mathbb{Z} \cup \{\infty, -\infty\}$.

The order $\leq$ works as usual. $-\infty$ is the least and ∞ the greatest element.

Addition becomes partial because $-\infty + \infty$ is undefined. We put $-\infty + z = -\infty$ for $z \neq \infty$.

Subtraction is introduced as usual. Thus, $z - \infty = -\infty - z = -\infty$ for $z \in \mathbb{Z}$. $\infty - \infty$ is undefined.

Multiplication works similarly to $\mathbb{N}^\infty$. $-\infty \cdot 0$ is undefined. And for $x \neq 0$, we define $\infty \cdot x$ and $-\infty \cdot x$ as ∞ or $-\infty$ depending on the signs.

1.6 Size of Sets

The size $|S|$ of a set S is a very complex operation because there are different degrees of infinity, i.e., not all infinite sets have the same size. Specifically, we have that $|\mathcal{P}(S)| > |S|$, i.e., we have infinitely many degrees of infinity.

In computer science, we are only interested in countable sets. Therefore, we use a much simpler definition of size: we write C for *countable* and U for uncountable, i.e., everything that is bigger:

Definition 1.56 (Size of sets)**.** The size $|S| \in \mathbb{N} \cup \{C, U\}$ of a set S is defined by:

- if S is finite: $|S|$ is the number of elements of S
- if S is infinite and bijective to $\mathbb{N}$: $|S| = C$, and we say that S is **countable**
- if S is infinite and not bijective to $\mathbb{N}$: $|S| = U$, and we say that S is **uncountable**

We can compute with set sizes as follows:

Definition 1.57 (Computing with Sizes)**.** For two sizes $s, t \in \mathbb{N} \cup \{C, U\}$, we define addition, multiplication, and exponentiation by the following tables:

	$s+t$	t: $n \in \mathbb{N}$	C	U
	$m \in \mathbb{N}$	$m+n$	C	U
s	C	C	C	U
	U	U	U	U

	$s*t$	t: $n \in \mathbb{N}$	C	U
	$m \in \mathbb{N}$	$m*n$	C	U
s	C	C	C	U
	U	U	U	U

	s^t	t: 0	1	$n \in \mathbb{N} \setminus \{0\}$	C	U
	0	1	0	0	0	0
	1	1	1	1	1	1
s	$m \in \mathbb{N} \setminus \{0\}$	1	m	m^n	U	U
	C	1	C	C	U	U
	U	1	U	U	U	U

Because exponentiation s^t is not commutative, the order matters: s is given by the row and t by the column.

The intuition behind these rules is given by the following:

Theorem 1.58. *For all sets S, T, we have for the size of the*

- *disjoint union:*
$$|S \uplus T| = |S| + |T|$$
- *Cartesian product:*
$$|S \times T| = |S| * |T|$$
- *set of functions from T to S:*
$$|S^T| = |S|^{|T|}$$

Thus, we can understand the rules for exponentiation as follows. Let us first consider the 4 cases where one of the arguments has size 0 or 1: For every set A

1. there is exactly one function from the empty set (namely the empty function): $|A^{\varnothing}| = 1$,
2. there are as many functions from a singleton set as there are elements of A: $|A^{\{x\}}| = |A|$,
3. there are no functions to the empty set (unless A is empty): $|\varnothing^A| = 0$ if $A \neq \varnothing$,
4. there is exactly one function into a singleton set (namely the constant function): $|\{x\}^A| = 1$,

Now we need only one more rule: The set of functions from a non-empty finite set to a finite/countable/uncountable set is again finite/countable/uncountable. In all other cases, the set of functions is uncountable.

1.7 Important Sets and Functions

The meaning and purpose of a data structure is to describe a set in the sense of mathematics. Similarly, the meaning and purpose of an algorithm is to describe a function between two sets.

Thus, it is helpful to collect some sets and functions as examples. These are typically among the first data structures and algorithms implemented in any programming language and they serve as test cases for evaluating our languages.

1.7.1 Base Sets

When building sets, we have to start somewhere with some sets that are assumed to exist. These are called the *bases sets* or the *primitive sets*.

The following table gives an overview of commonly used base sets, where we also list the size of each set according to Def. 1.56:

set	description/definition	size
typical base sets of mathematics[7]		
$\varnothing$	empty set	0
$\mathbb{N}$	natural numbers	C
$\mathbb{Z}$	integers	C
$\mathbb{Z}_m$ for $m > 0$	integers modulo m, $\{0, \ldots, m-1\}$ [8]	m
$\mathbb{Q}$	rational numbers	C
$\mathbb{R}$	real numbers	U
additional or alternative base sets used in computer science		
void	alternative name for $\varnothing$	0
unit	unit type, $\{()\}$, equivalent to $\mathbb{Z}_1$	1
$\mathbb{B}$	booleans, $\{\mathit{false}, \mathit{true}\}$, equivalent to $\mathbb{Z}_2$	2
int	primitive integers, $-2^{n-1}, \ldots, 2^{n-1}-1$ for machine-dependent n, equivalent to $\mathbb{Z}_{2^n}$ [9]	2^n
float	IEEE floating point approximations of real numbers	C
char	characters	finite[10]
string	lists of characters	C

1.7.2 Functions on the Base Sets

For every base set, we can define some basic operations. These are usually built-in features of programming languages whenever the respective base set is built-in.

We only list a few examples here.

Numbers

For all number sets, we can define addition, subtraction, multiplication, and division in the usual way.

Some care must be taken when subtracting or dividing because the result may be in a different set. For example, the difference of two natural numbers is not in general a natural number but only an integer (e.g., $3 - 5 \notin \mathbb{N}$). Moreover, division by 0 is always forbidden.

Quotients of the Integers

The function $\mathit{modulus}_m$ (see Sect. 1.5.3) for $m \in \mathbb{N}$ maps $x \in \mathbb{Z}$ to $x \bmod m \in \mathbb{Z}_m$.

In programming languages, the set $\mathbb{Z}_m$ is usually not provided. Instead, $x \bmod y$ is built-in as a function on *int*. [11]

Booleans

On booleans, we can define the usual boolean operations conjunction (usually written & or &&), disjunction (usually written | or ||), and negation (usually written !).

Moreover, we have the equality and inequality functions, which take two objects x, y and return a boolean. These are usually written $x == y$ and $x \mathrel{!}= y$ in text files and $x = y$ and $x \neq y$ on paper.

1.7.3 Set Constructors

From the base sets, we build all other sets by applying set constructors. Those are operations that take sets and return new sets.

[7] All of mathematics can be built by using $\varnothing$ as the only base set because the others are definable. But it is common to assume at least the number sets as primitives.

[8] $\mathbb{Z}_0$ also exists but is trivial: $\mathbb{Z}_0 = \mathbb{Z}$.

[9] Primitive integers are the 2^n possible values for a sequence of n bits. Old machines used $n = 8$ (and the integers were called "bytes"), later machines used $n = 16$ (called "words"). Modern machines typically use 32-bit or 64-bit integers. Modern programmers usually—but dangerously—assume that 2^n is much bigger than any number that comes up in practice so that essentially (but not actually) $\mathit{int} = \mathbb{Z}$. Some programming languages (e.g., Python) correctly implement $\mathit{int} = \mathbb{Z}$.

[10] The ASCII standard defined 2^7 or 2^8 characters. Nowadays, we use Unicode characters, which is a constantly growing set containing the characters of virtually any writing system, many scientific symbols, emojis, etc. Many programming languages assume that there is one character for every primitive integers, e.g., typically 2^{32} characters.

[11] Some care must be taken if x is negative because not all programming languages agree.

The following table gives an overview of commonly used set constructors, where we also list the size of each set according to Def. 1.57:

set	description/definition	size
typical constructors in mathematics		
$A \uplus B$	disjoint union	$\|A\| + \|B\|$
$A \times B$	(Cartesian) product	$\|A\| * \|B\|$
A^n for $n \in \mathbb{N}$	n-dimensional vectors over A	$\|A\|^n$
B^A or $A \to B$	functions from A to B	$\|B\|^{\|A\|}$
$\mathcal{P}(A)$	power set, equivalent to $\mathbb{B}^A$	$2^{\|A\|} = \begin{cases} 2^n & \text{if } \|A\| = n \\ U & \text{otherwise} \end{cases}$
$\{x \in A \| P(x)\}$	subset of A given by property P	$\leq \|A\|$
$\{f(x) : x \in A\}$	image of operation f when applied to elements of A	$\leq \|A\|$
A/r	quotient set for an equivalence relation r on A	$\leq \|A\|$
selected additional constructors often used in computer science		
A^*	lists over A	$\begin{cases} 1 & \text{if } \|A\| = 0 \\ U & \text{if } \|A\| = U \\ C & \text{otherwise} \end{cases}$
$A^?$	optional element[12] of A	$1 + \|A\|$
for new names $l_1, \ldots, l_n$		
$enum\{l_1, \ldots, l_n\}$	enumeration: like $\mathbb{Z}_n$ but also introduces named elements l_i of the enumeration	n
$l_1(A_1)\|\ldots\|l_n(A_n)$	labeled union: like $A_1 \uplus \ldots \uplus A_n$ but also introduces named injections l_i from A_i into the union	$\|A_1\| + \ldots + \|A_n\|$
$\{l_1 : A_1, \ldots, l_n : A_n\}$	record: like $A_1 \times \ldots \times A_n$ but also introduces named projections l_i from the record into A_i	$\|A_1\| * \ldots * \|A_n\|$
inductive data types[13]		C
classes[14]		U

1.7.4 Characteristic Functions of the Set Constructors

Every set constructor comes systematically with characteristic functions into and out of the constructed sets S. These functions allow building elements of S or using elements of S for other computations.

For some sets, these functions do not have standard notations in mathematics. In those cases, different programming languages may use slightly different notations.

The following table gives an overview:

set C	build an element of C	use an element x of C
$A_1 \uplus A_2$	$inj_1(a_1)$ or $inj_2(a_2)$ for $a_i \in A_i$	pattern-matching
$A_1 \times A_2$	(a_1, a_2) for $a_i \in A_i$	$x.i \in A_i$ for $i = 1, 2$
A^n	$(a_1, \ldots, a_n)$ for $a_i \in A$	$x.i \in A$ for $i = 1, \ldots, n$
B^A	$(a \in A) \mapsto b(a)$	$x(a)$ for $a \in A$
A^*	$[a_0, \ldots, a_{l-1}]$[15] for $a_i \in A$	pattern-matching
$A^?$	$None$ or $Some(a)$ for $a \in A$	pattern-matching
$enum\{l_1, \ldots, l_n\}$	l_1 or …or l_n	switch statement or pattern-matching
$l_1(A_1)\|\ldots\|l_n(A_n)$	$l_1(a_1)$ or …or $l_n(a_n)$ for $a_i \in A_i$	pattern-matching
$\{l_1 : A_1, \ldots, l_n : A_n\}$	$\{l_1 = a_1, \ldots, l_n = a_n\}$ for $a_i \in A_i$	$x.l_i \in A_i$
inductive data type A	$l(u_1, \ldots, u_n)$ for a constructor l of A	pattern-matching
class A	**new** A	$x.l(u_1, \ldots, u_n)$ for a field l of A

[12]An optional element of A is either absent or an element of A.

[13]These are too complex to define at this point. They are a key feature of functional programming languages like SML.

[14]These are too complex to define at this point. They are a key feature of object-oriented programming languages like Java.

[15]Mathematicians start counting at 1 and would usually write a list of length n as $[a_1, \ldots, a_n]$. However, computer scientists always start counting at 0 and therefore write it as $[a_0, \ldots, a_{n-1}]$. We use the computer science numbering here.