Chapter I — §1 Monoids

Motivation-forward slides: what, why, and where it matters

Slides generated from your chapter outline (no exercises)

August 20, 2025

Section roadmap

Why start with monoids?

Why monoids first?

- **Minimal algebra of composition:** the least structure that lets you *combine* results repeatedly.
- **Universal base case:** sums, products, function composition, string concatenation—each is (at least) a monoid.
- **Bridge to everything else:** add inverses $\Rightarrow$ groups; add a second operation $\Rightarrow$ rings/semirings; view as one-object categories.
- **Practical pay-off:** associative $\oplus$ + identity e gives safe *fold/reduce*, parallelization, and incremental computation.

Before superheroes (groups) come the capes and boots (monoids).

Binary laws of composition

What is a law of composition?

- A **law of composition** on a set S is a map

$$\mu : S \times S \to S, \qquad (x, y) \mapsto x \cdot y.$$

- Often write xy for $x \cdot y$; if commutative, use $x + y$.

Why this matters
Saying *"closed under combining"* is how we guarantee that iterative processes never leave the universe we care about.

Associativity at center stage

- **Associative** means $(xy)z = x(yz)$ for all x, y, z.
- Convention: the **empty product** equals the unit e (when a unit exists).

Why associativity is king

- *Parenthesis-free* evaluation: $x_1 x_2 \cdots x_n$ is unambiguous.
- *Parallelism*: chunk-then-merge yields the same answer (MapReduce vibes).
- *Induction/folds*: proofs and programs can process streams incrementally.

Parentheses are like seatbelts: you only notice them when something non-associative happens.

Definition and basic properties

Definition: monoid

Monoid

A **monoid** is a triple $(M, \cdot, e)$ where M is a set, $\cdot$ is associative, and e is a two-sided unit: $ex = xe = x$ for all $x \in M$.

- If $xy = yx$ for all x, y, the monoid is **commutative** (often written $(M, +, 0)$).
- Elements with two-sided inverses are **units**; these form a group $M^\times$.

Why the unit matters

The unit is the *do-nothing* element: the base case for recursion, streaming, and identity effects in composition.

Uniqueness of the unit (blink-and-you-miss-it)

Proposition
If e and e' are both units in M, then $e = e'$.

Proof
$e = e \cdot e' = e'$.

Why this matters
There is a *single* neutral baseline in which to start or end computations; your folds don't depend on which "identity" you picked.

In monoids, the identity is strictly monogamous.

Left/right units and inverse uniqueness

- Left unit: $ex = x$ for all x; right unit: $xe = x$ for all x.
- If both exist (with associativity), they coincide.
- If $xu = ux = e$ and $xv = vx = e$, then $u = v$ (inverse uniqueness).

Why this matters

One coherent "neutral" behavior simplifies algebraic manipulations and program laws—no special-casing left vs. right.

Two-sided inverses: because who wants commitment only on weekdays?

Powers and exponent laws

Let $(M, \cdot, e)$ be a monoid and $x \in M$.

- $x^0 := e$, $x^{n+1} := x^n x$; then $x^{m+n} = x^m x^n$ and $(x^m)^n = x^{mn}$.
- If $xy = yx$, then $(xy)^n = x^n y^n$.

Why this matters
These give compact algebra for iterated composition—think "apply a transformation n times" or "aggregate n records."

Your high-school exponent rules? Monoid lore in disguise.

Examples and non-examples

Classic examples (where monoids live)

- $(\mathbb{N}, +, 0)$, $(\mathbb{Z}, +, 0)$; $(\mathbb{N}, \times, 1)$.
- $M_n(R)$ with matrix multiplication and I_n.
- $\mathrm{End}(S)$: all $S \to S$ under composition with id_S.
- Strings Σ^* under concatenation; unit ε.
- Idempotent monoids: $(\mathbb{R}_{\geq 0}, \max, 0)$, Boolean $(\{0, 1\}, \vee, 0)$.
- Logs/metrics: combine by sum, max, or concatenation.

Why these matter
They power *folds*, *dynamic programming*, and *parallel reductions* in real workloads.

Non-examples & boundaries

- $(\mathbb{R}, -, 0)$ is not associative.
- Singular matrices $\not\ni I \Rightarrow$ no unit.

Why boundaries matter
Knowing where axioms *fail* prevents silent bugs (e.g., trying to parallelize a non-associative reduction).

If it won't associate, it won't cooperate.

Submonoids and generation

Submonoids

Definition
$N \subseteq M$ is a **submonoid** if $e \in N$ and $xy \in N$ whenever $x, y \in N$.

Why this matters
They are the *stable subsystems* under composition—useful for invariants and restricting attention to feasible states.

Generated submonoids

Definition
Given $S \subseteq M$, the **submonoid generated by** S, $\langle S \rangle$, is the intersection of all submonoids containing S.

- Concretely: all finite products of elements of S (empty product allowed).

Why this matters
Lets us *build* from primitives and reason about expressiveness: which behaviors are achievable from a chosen toolkit?

From parts list to full kit—LEGO algebra.

Units, cancellation, idempotents

Group of units

- $u \in M$ is a **unit** if some v satisfies $uv = vu = e$.
- Units form a group $M^\times$.

Why this matters
Units capture *reversible* transformations hiding inside a possibly irreversible world—vital in algorithm design and simplification.

Cancellation vs. invertibility

- Left-cancellative: $ax = ay \Rightarrow x = y$; right-cancellative: $xa = ya \Rightarrow x = y$.
- Units imply cancellation; not conversely in general monoids.

Why this matters
Cancellation is the algebraic form of “no information lost” when composing with a—useful for uniqueness and injectivity arguments.

Being cancellative is like being persuasive; having an inverse is like having receipts.

Idempotents and absorbing elements

- Idempotent: $p^2 = p$. Absorbing: $0x = x0 = 0$.
- In idempotent commutative monoids (join-semilattices), $x + y$ models union/OR.

Why this matters

Idempotents model *stabilization* and fixed points; absorbing elements model *fail-fast* behavior (once zero, always zero).

Finite products and indexing

Products over finite index sets

- If only finitely many $x_i \neq e$, define $\prod_{i \in I} x_i$ safely.
- For finitely supported $f : I \times J \to M$,

$$\prod_{i \in I} \prod_{j \in J} f(i,j) = \prod_{(i,j) \in I \times J} f(i,j) = \prod_{j \in J} \prod_{i \in I} f(i,j).$$

Why this matters
Reindexing arguments are the backbone of many combinatorial identities and correctness proofs for parallel aggregation.

Reindex responsibly. Associativity is your seatbelt; commutativity is cruise control.

Morphisms and quotients

Monoid homomorphisms

Definition
$f : (M, \cdot, e) \to (N, \star, 1)$ with $f(x \cdot y) = f(x) \star f(y)$ and $f(e) = 1$.

Why this matters
Homomorphisms are the *structure-preserving* maps—reuse computations, transport properties, and compare models.

Congruences and first isomorphism theorem

- A **congruence** $\sim$ respects multiplication: $x \sim x'$, $y \sim y' \Rightarrow xy \sim x'y'$.
- Quotient $M/{\sim}$ is a monoid; kernel congruence of f yields $M/{\sim} \cong \mathrm{Im}(f)$.

Why this matters
Quotients *identify indistinguishable states*: minimize automata, compress logs, or factor out harmless details.

Same heist as in group theory, different getaway car.

Free monoids and presentations

Free monoids

- For an alphabet Σ, Σ^* (all finite words) under concatenation; unit ε.
- **Universal property:** any $g : \Sigma \to M$ extends uniquely to $\widehat{g} : \Sigma^* \to M$.

Why this matters

This turns *syntax* (words) into *semantics* (elements) in one shot; it's the engine behind substitution and evaluation.

Presentations

- $M \cong \Sigma^*/\equiv$ with relations generating a smallest congruence.
- Example: commutative monoid on x, y is $\langle x, y \mid xy = yx \rangle$.

Why this matters

Presentations let us describe *huge* structures economically and prove properties by rewriting.

Writing down every element one by one is a terrible hobby.

Actions and applications

Monoid actions

Definition
An action of $(M, \cdot, e)$ on S is a map $M \times S \to S$ with $e \cdot s = s$ and $x \cdot (y \cdot s) = (xy) \cdot s$.

- Equivalently: a homomorphism $M \to \mathrm{End}(S)$.
- Example: $\mathbb{N}$ acts by iterates of a function $f : S \to S$.

Why this matters
Actions model *processes over states*: iterating transformations, scheduling effects, or running automata.

When monoids stop being polite and start acting (on sets).

Checklists and pitfalls

Monoid verification checklist

1. Specify the set M and the operation $\cdot$.
2. Prove associativity clearly.
3. Exhibit a two-sided unit e.
4. Identify $M^{\times}$, notable submonoids, natural homomorphisms.

Why this matters
A clean checklist prevents "almost a monoid" mistakes that break folds, proofs, or parallelization.

Common pitfalls

- Assuming left identity implies right identity *without* associativity.
- Using cancellation where invertibility (or cancellativity) isn't guaranteed.
- Forgetting the empty product convention in product manipulations.

Micro–summary
Monoids $=$ associative composition $+$ identity. That's enough to power folds, rebracketing, quotients, actions, and lots of real math.

If every element becomes a unit—welcome to **Groups**. *DLC unlocked.*