MACHINES, MONOIDS, CATEGORIES
SUPPLEMENTAL MATERIAL FOR UW PLSE CATEGORY THEORY SUMMER 2023 CLASS (INSTRUCTOR: JOHN LEO)

GILBERT LOUIS BERNSTEIN

In class, John Leo described monoids as "like groups, but more relevant to computer scientists, because we don't always have inverses." This note attempts to make that observation more explicit by connecting monoids to (finite) state-machines. This note asks you to think about whether these are "the same" or not, and how exactly they are connected.

State machines (far beyond merely *finite* state machines) are the basis for most computing theory. Turing machines are state machines. When we define a small-step operational semantics for a programming language, we define a state machine. Any hardware design implements a finite state machine. Lexing and parsing is built on finite and pushdown state machines. Model checking verifies properties of finite state machines. And so on.

Why bother working this out? You may only have a passing interest in the usual mathematical examples of algebraic structures. (e.g. group, ring, field, vector-space) By contrast, you will almost certainly find productive connections to monoids in your current or future research.

1. (FINITE) STATE MACHINES: REVIEW

Definition 1.1. Recall that we can define a deterministic state machine as a tuple $SM = (Q, A, q_0, \delta)$ representing:

- Q a set of states
- A an alphabet of symbols
- $q_0 \in Q$ an initial state
- $\delta : Q \times A \to Q$ the transition function, which given an input symbol $a \in A$ and a current state q produces the next state $\delta(q, a)$

If the set Q is finite, we say that SM is a **finite** state machine. If we replace the function δ by a relation $\delta \subseteq (Q \times A) \times Q$, or equivalently[1] by a function

[1]Why? Recall that the powerset of a set A is defined as $\mathcal{P}(A) = \{X | X \subseteq A\}$. Now, consider the type $A \to \mathsf{Bool}$ interpreted as the set of all functions from A to Bool. There exists a bijection between these two sets. For $X \subseteq A$, define $\mathbf{1}_X(x) = (x \in X)$ sometimes called the "characteristic function" or "indicator function" of X; for $f : A \to \mathsf{Bool}$, define

$\delta : Q \times A \to \mathcal{P}(Q)$ (i.e. producing a set of possible next states) then we say that SM is **non-deterministic**.

At this point, I am supposed to draw a picture of a graph with little circles and arrows between them, and each arrow labeled with a symbol from the alphabet A. I'm lazy and want to keep typing, so let me give you a different example.

Example 1.2 (n-bit counter). Let Q be the set of all n-bit non-negative integers. Let $A = a$ be an alphabet with only one symbol. Let $q_0 = 0$. Finally, for all $q < 2^{n-1}$, let $\delta(q, a) = q + 1$ and $\delta(2^{n-1}, a) = 0$. This finite state machine represents an n-bit counter. It doesn't do much, but you're going to get really exhausted trying to draw it for $n = 32$.

Example 1.3 (n-bit counter with on/off control). Let Q and q_0 be the same as before, but $A = \{0, 1\}$. Let $\delta(q, 1) = \delta(q, a)$ from the last example, but let $\delta(q, 0) = q$.

2. Monoids

Recall the definition of **monoid** from John's lecture.

Definition 2.1. A monoid is a triple $(M, \cdot, 1)$ where

- X is a set, (sometimes called a "carrier set")
- $\cdot : M \times M \to M$ is the "product" operation between elements of the monoid
- $1 \in M$ is the "unit" a distinguished element of the monoid
- associativity: $\forall x, y, z \in M, (x \cdot y) \cdot z = x \cdot (y \cdot z)$
- identity: $\forall x \in M, 1 \cdot x = x \cdot 1 = x$

We often merely refer to M as the monoid, when the choice of product and unit are unambiguous.

Here are two examples of monoids.

Example 2.2 (natural numbers under addition). One example of a monoid we saw in class is $(\mathbb{N}, +, 0)$, the natural numbers, with addition as the monoidal product and 0 as the monoidal unit.

$\text{supp}_f = \{x | f(x) = 1\}$. One can check that these two maps are inverses. Thus a bijection is demonstrated. By combining this equivalence with Currying, we get

$$\begin{aligned} Q \times A \to \mathcal{P}(Q) &\cong Q \times A \to Q \to \mathsf{Bool} \\ &\cong Q \times A \times Q \to \mathsf{Bool} \\ &\cong \mathcal{P}(Q \times A \times Q) \end{aligned}$$

Example 2.3 (integers under addition). We also observed that $(\mathbb{Z}, +, 0)$, is a monoid.

Example 2.4 (integers modulo n under addition). Finally, we observed that $(\mathbb{Z}_n, +, 0)$ is also a monoid, when $\mathbb{Z}_n$ is the integers modulo n, and addition is performed modulo n.

Have you ever read math notes where the examples seem like they're very suggestively chosen? I wonder why authors do that.

3. GENERATORS

Our examples of monoids were in some sense overly simple. On the other hand, they concerned both infinite or arbitrarily large carrier sets. Generators can help us get a grasp on why these examples were so simple to describe.

Definition 3.1 (generators of a monoid)**.** Let M be a monoid, and let $G \subseteq M$ be any subset of the monoid. If every element $x \in M$ can be written as a product $g_1 \cdot g_2 \cdots g_n$ of 0 or more elements $g_i \in G$ (repetition allowed) then we say that G are a set of *generators* for M. (In the case where we write x as a product of zero generators, we mean that $x = 1_M$, the unit)

Lemma 3.2 (a trivial generating set)**.** *Let M be a monoid. Then M is a generating set for M*

Proof. Every element $x \in M$ can be written as the word x. □

Problem 3.3 (generating sets for $\mathbb{N}$). Is $\{0\}$ a generating set for $\mathbb{N}$? Is $\{1\}$ a generating set for $\mathbb{N}$? Is $\{2, 3\}$ a generating set for $\mathbb{N}$? Give proofs or counterexamples for each question.

Problem 3.4 (generating sets for $\mathbb{Z}$). Is $\{0\}$ a generating set for $\mathbb{Z}$? Is $\{1\}$ a generating set for $\mathbb{Z}$? Is $\{-1, 1\}$ a generating set for $\mathbb{Z}$? Is $\{2, 3\}$ a generating set for $\mathbb{Z}$? Is $\{-2, 3\}$ a generating set for $\mathbb{Z}$? Give proofs or counterexamples for each question.

Problem 3.5 (generating sets for $\mathbb{Z}_n$). Is $\{0\}$ a generating set for $\mathbb{Z}_n$? Is $\{1\}$ a generating set for $\mathbb{Z}_n$? Is $\{2, 3\}$ a generating set for $\mathbb{Z}$? Give proofs or counterexamples for each question.

If you've taken number theory, you may also enjoy thinking about this problem. But if not, you should probably skip it.

Problem 3.6 (which elements generate $\mathbb{Z}_n$?). Consider any $k \in \mathbb{Z}_n$. Under what conditions is $\{k\}$ a generator for $\mathbb{Z}_n$?

Up to this point, we considered identifying generating sets in already defined monoids. What if we took an arbitrary set and tried to define a monoid from it?

Definition 3.7 (the Free Monoid)**.** The free monoid on a set G consists of all strings (also called "words") formed by sequences of zero or more elements from G (allowing repetition). We write ϵ to mean the empty sequence, and G^* to mean the infinite set of such strings. Then, G^* forms a monoid with $x_1 \cdot x_2 = x_1 x_2$ defined as string concatenation and ϵ as the unit. One can easily verify that this satisfies the monoid axioms.

4. Machines or Monoids?

I will provide one obvious connection between monoids and state machines. Then, I will ask you to prove a straightforward extension of this idea. Finally, I will pose a question that's harder to get right.

Monoids and state machines have some suggestive connections. For example, state machines take strings as input, and elements of monoids can be described using strings of generators. State machines have sets of states, and monoids have carrier sets. But in a monoid, any two elements can be multiplied. And in a state machine, one must have defined an alphabet.

Definition 4.1 (Largest Alphabet Machine)**.** Let $(M, \cdot, 1)$ be a monoid. Then, define the state machine $\text{MaxMachine}(M) = (M, M, 1, \cdot)$ with M as both the set of states and the alphabet, 1 as the initial state, and with $\cdot : M \times M \to M$ as the transition function. Then $\text{MaxMachine}(M)$ is a state machine. If M is finite, then so is the machine.

Corollary 4.2 (Computing Equivalence)**.** *Let M be a (finite) monoid. And let $x, y \in M^*$ be two strings. We can compute whether $x = y$ or not by running* $\text{MaxMachine}(M)$ *on both x and y as inputs and seeing whether they end up in the same state or not.*

Problem 4.3. We just showed a way to construct a state machine from an arbitrary monoid. Can you give an example of a state machine SM for which there does not exist any monoid M s.t. $\text{MaxMachine}(M) = SM$?

One concern might be that most state machines have alphabets that are smaller than the number of states of the machine. (challenge: if $Q = A$ can you still solve the preceding problem?) Can we construct a similar machine with the same computational utility as the "Largest Alphabet" machine defined above, but using a smaller alphabet?

Problem 4.4 (Machine given Generators). Can you construct a state machine from the combination of a monoid M and generating set G?

You should be able to provide the preceding construction without too much trouble. However, now consider the opposite direction. Can we construct a monoid from a given (finite) state machine? The situation appears to be more complicated.

Example 4.5 (A Possible Obstruction). Consider a finite state machine with 4 states $Q = \{1, 2, 4_a, 4_b\}$, a 2 character alphabet $A = \{a, b\}$, initial state

$q_0 = 1$, and transition function given by the rules $\delta(1, -) = 2$, $\delta(2, a) = 4_a$, $\delta(2, b) = 4_b$, and $\delta(4_i, -) = 4_i$. We might be tempted to try to construct a monoid using $\{a, b\}$ as a generating set. However, we run into a problem. If $a = b$ in the monoid, then wouldn't $aa = bb$ also be true in the monoid? But in the state machine, those two input strings take us to different states. Alternately, if we said $a \neq b$ in the monoid, then we would have the problem that input strings a and b take us to the same state in the state machine.

Problem 4.6. Can you come up with a way to resolve this problem? Can you define one mapping from machines to monoids and then a second mapping from monoids to machines (maybe one of the two options above, but not necessarily) such that the two mappings are inverses of each other?