Contents

Contents 1

A. E PLURIBUS 1

1. Compositionality [draft] 3
1.1. Putting things together [draft] . . . 3
1.2. Examples for Andrea . . . 7

2. E pluribus unum [draft] 11
2.1. Semigroups [draft] . . . 11
Induced n-ary multiplication . . . 13
Code exercises . . . 14
2.2. Monoids [draft] . . . 14
Code exercises . . . 17
2.3. Rope Goldberg Machines . . . 18
2.4. Generators and relations [draft] . . . 22
2.5. Groups [draft] . . . 23

3. Homomorphisms and actions [draft] 25
3.1. Homomorphisms . . . 25
3.2. Monoid homomorphism [missing] . . . 28
3.3. Group homomorphism . . . 29
3.4. Actions [missing] . . . 29

4. Solutions to selected exercises 31

E pluribus | Part A.

(short desc)
(caption)long desc

1. **Compositionality** [draft] **3**
2. **E pluribus unum** [draft] **11**
3. **Homomorphisms and actions** [draft] **25**
4. **Solutions to selected exercises** **31**

1. Compositionality [draft]

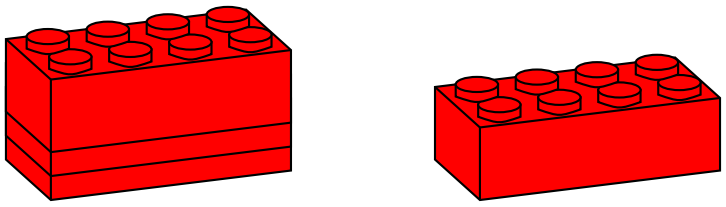
In this chapter we discuss the various types of "compositions" we find in applications.

1.1 Putting things together [draft] . 3
1.2 Examples for Andrea 7

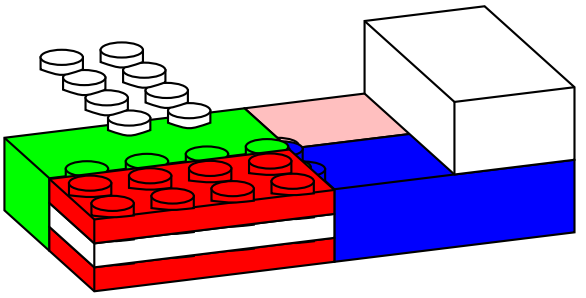
1.1. Putting things together [draft]

Lego The first encounter children have with composition is with toy blocks like Lego. It is a coincidence that there is a *lego* in *intellego*; the lego in Lego is a contraction from Danish *leg godt*, which means *to play well*.

Legos are compositional in this sense: when you put together two blocks, you can treat the ensemble as one block for the purpose of composing it with other blocks. For example, you can take 3 blocks of dimension $1/3 \times 2 \times 4$ and compose them together to obtain one block of dimension $1 \times 2 \times 4$. This composed block works the same as one primitive $1 \times 2 \times 4$ block.



The city of Lucerne made of Lego at Legoland, Billund, Denmark.



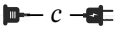
Here is one formal property of Lego: given two blocks of the same shape, you can always make a block of twice the shape.

More generally, to compose blocks in this way, you would only care about the lateral dimensions. You can compose an $a \times b \times c$ block with an $a \times b \times d$ block to obtain an $a \times b \times (c + d)$ block. Note that, for now, a and b are treated as labels, not as numbers.

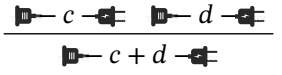
We like to put this in formula as follows. We put the ingredients to the top, and the results at the bottom.

$$\frac{\text{block } a \times b \times c \quad \text{block } a \times b \times d}{\text{block } a \times b \times (c + d)} \tag{1.1}$$

Extension cords Do you know the game "spot the 5 differences"? In this book we are going to play the opposite game, which is "spot how different things are the same at some level of abstraction".

It might have been a while since you played with Lego, but you are certainly familiar with plugs, sockets, and extension cords. Let  be an extension cord of length c.

If you have an extension cord of length c and another of length d, you can plug them together to get an extension cord of length $c + d$.

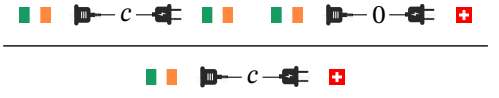
 (1.2)

The rule for extension cords is similar to the rule for Lego blocks.

Extension cords are a bit more general: Lego blocks are constrainted to have a height that is a multiple of $1/3$ of brick, but extension cords can have any continuous value as length.

On the other hand, Lego blocks also have this other property of the horizontal section $a \times b$. e only gave rules for the connection of blocks of the same horizontal section. What would be the equivalent for extension cords?

As you read this book and start to plan to visit Switzerland, at some point you need to buy some adapters. Switzerland uses the connector of type N (Fig. 1.2l). If you come from Ireland, your appliances use type G (Fig. 1.2g).

 (1.3)

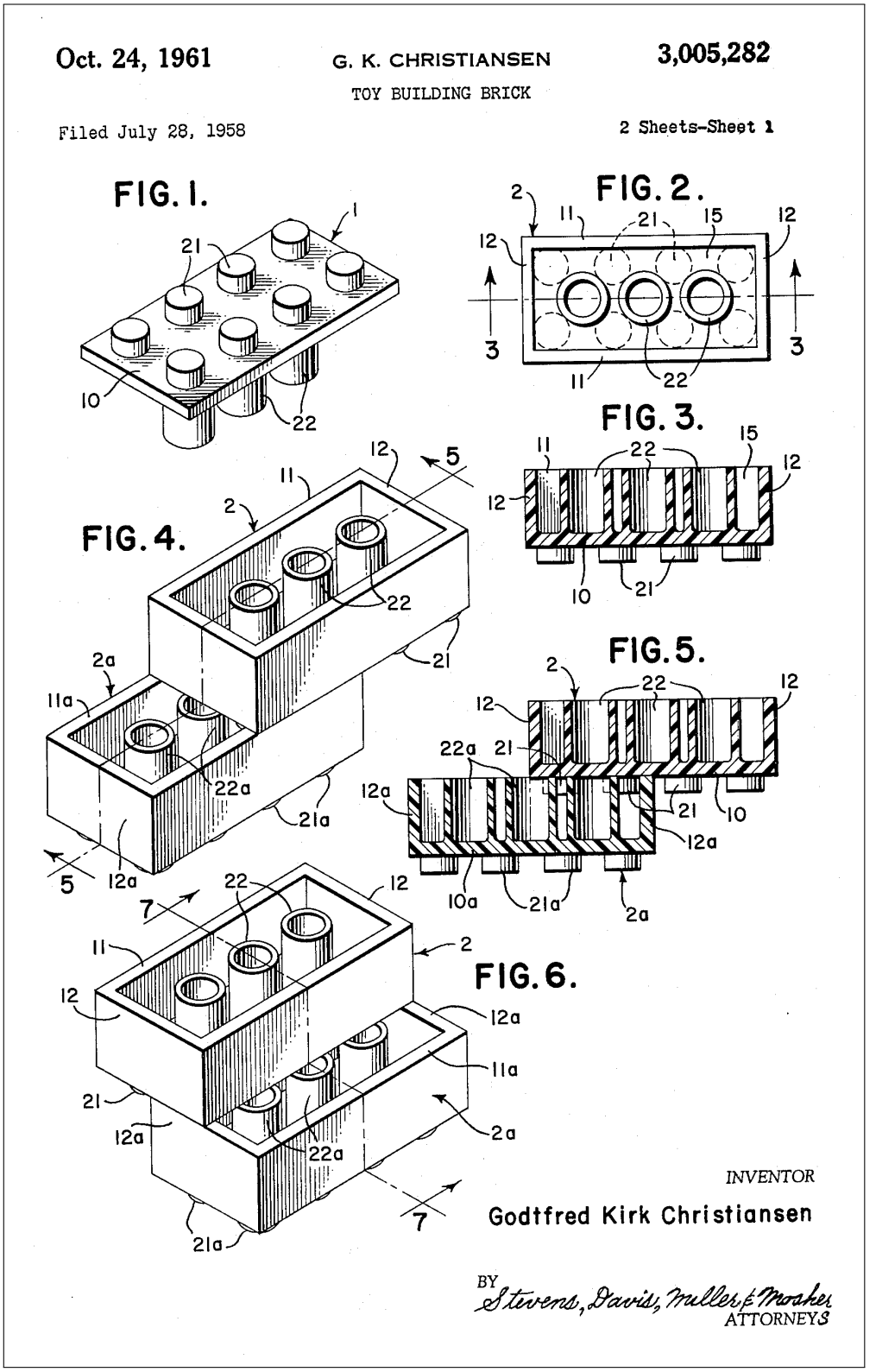
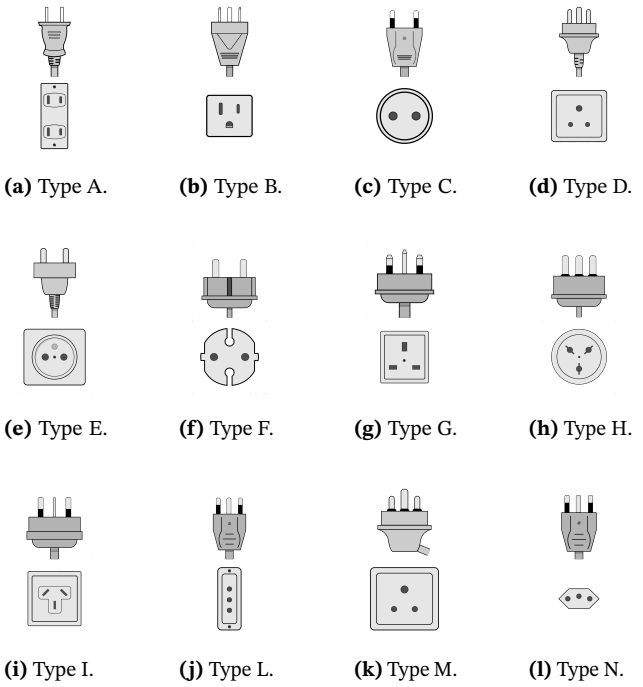


Figure 1.1.: The 1961 Lego patent.



(a) Type A. **(b)** Type B. **(c)** Type C. **(d)** Type D.

(e) Type E. **(f)** Type F. **(g)** Type G. **(h)** Type H.

(i) Type I. **(j)** Type L. **(k)** Type M. **(l)** Type N.

Figure 1.2.: Plug/socket types used in the world

Sonia is making us customized icons without branding (now seen in plugs)

Lego with colors If you have legos of different colors, it becomes a bit more complicated. The rule now becomes

lego figure here: legos with colors

Lego operations There are also other ways you could compose the 3 blocks. You could slightly translate them, to create stairs.

todographicslego figure here: the 3 blocks composed in different ways, by some offset

You can also rotate them before composing.

lego figure here: the 3 blocks composed using rotations

What are the rules of this composition? Let's define a block as a solidly connected set of parallelipipedes (malformed cubes with different dimensions of the 3 sides). There are 3 kinds of faces: the faces pointing up, which have the pins, the faces pointing down, which have the holes for the pins, and the lateral faces.

Connecting two blocks means that there should be at least one pin face of the lower block touching one hole face of the upper block; and, that there are no intersections of the solid blocks. Then, the "interface" of the blocks

You can also put decorations; but if you do, you remove the possibility of connecting to all of them.

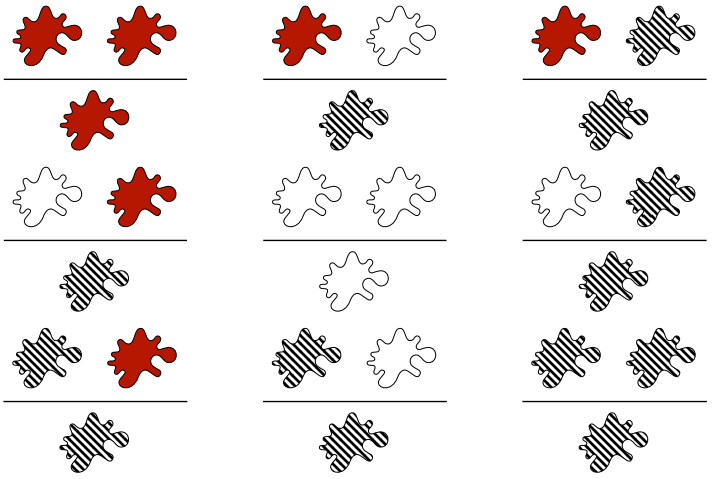
lego block with some decoration on top

1.2. Examples for Andrea

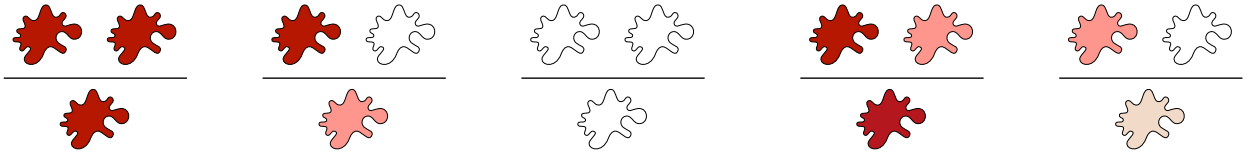
Example of a stain with possibility of choosing the color:

Example of a stain with a pattern of lines filling it:

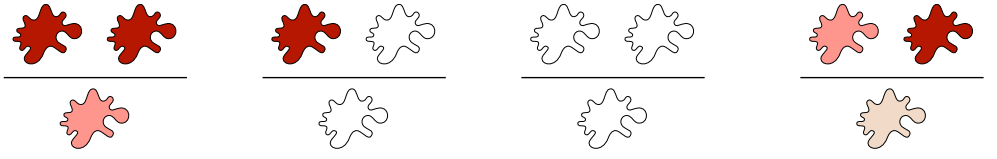
Example of sequents for slide 6:



Example of sequents for slide 7:

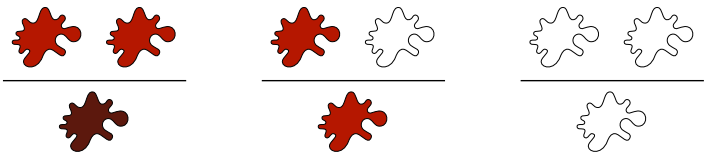


Example of sequents for slide 8:

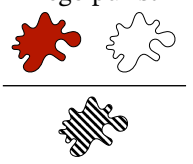
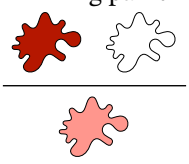
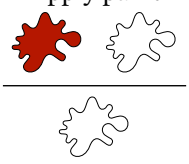
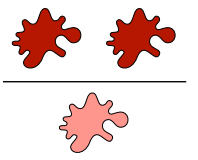
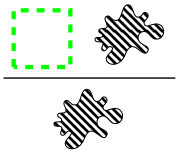
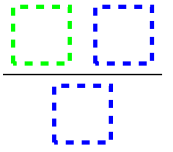
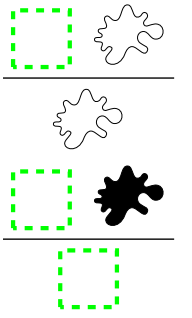


slide6
slide7
slide8

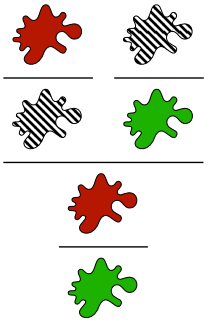
Example of sequents for slide 9:



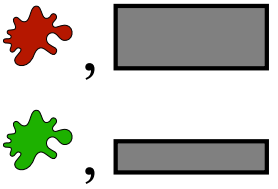
Example of sequents for slide 10:

	Lego purist	Mixing paint	Apply paint	Additive
				
Cumulative?	yes	yes	no	yes
Associative?	yes	no	yes	yes
				
0		no		
1	no	no	no	

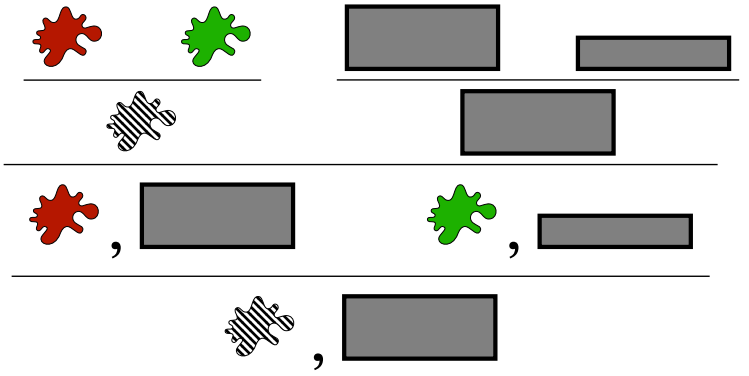
Example from slide 12:



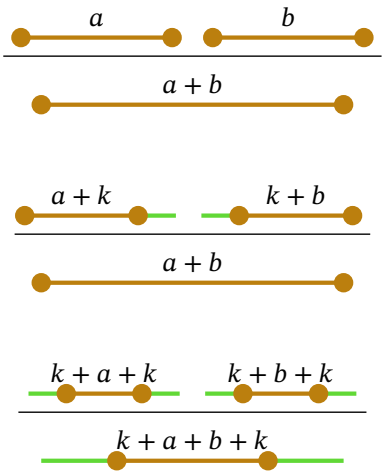
Example from slide 17:



slide9
slide10



Slide 21



2. E pluribus unum [draft]

This chapter introduces three of the most basic algebra structures: semigroups, monoids, and groups.

2.1 Semigroups [draft] 11
Induced n-ary multiplication . . 13
Code exercises 14
2.2 Monoids [draft] 14
Code exercises 17
2.3 Rope Goldberg Machines 18
2.4 Generators and relations [draft] 22
2.5 Groups [draft] 23

Oftentimes we are going to studies certain *structures* and then their *refinements*. By *refinement* we mean another type of structure that has additional properties/-constraints.

Consider a set of ...

2.1. Semigroups [draft]

Definition 2.1 (Semigroup). A *semigroup* $\mathbf{S}$ is a set S, together with a binary operation

$$\fatsemi : S \times S \to S, \tag{2.1}$$

called *composition*, which satisfies the *associative* law:

$$(x \fatsemi y) \fatsemi z = x \fatsemi (y \fatsemi z) \tag{2.2}$$

for all $x, y, z \in S$.

Remark. Given a fixed set **S**, there will in general be many different choices of composition operation which make **S** into a semigroup. So, technically, a semigroup is a pair $\langle \mathbf{S}, \fatsemi \rangle$ consisting of a set **S** and a choice of composition $\fatsemi$ *A*. The set **S** is the *underlying set* of the semigroup. Often we will be slightly imprecise and refer to a semigroup simply by the name of its underlying set; this is practical when it is clear from context which multiplication operation we are considering, or when it is not necessary to refer to the multiplication explicitly.

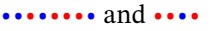
Example 2.2. Consider $\langle \mathbb{N}, + \rangle$. This is a semigroup, since, for all $l, m, n \in \mathbb{N}$, we have

$$(l + m) + n = l + (m + n).$$

Example 2.3. Consider a finite set *A*, which we think of as an alphabet. For instance, consider $A = \{\bullet, \bullet\}$. Let **S** be the set of non-empty strings of elements of *A*. For example,

is a non-empty string of elements of *A*.

We may define a multiplication operation on **S** simply by concatenating strings. Given the strings



their concatenation

is the string

It is readily seen that concatenation satisfies the associative law, so **S**, together with this multiplication, forms a semigroup.

When the underlying set of a semigroup $\langle \mathbf{S}, \fatsemi \rangle$ is a finite set, one way to specify the multiplication $\fatsemi$ is simply by writing out what it does with each pair of elements of **S**. Since $\fatsemi$ is a function of two variable, this can be conveniently displayed as a table, called a *multiplication table*.

JL: write out (and include graphic) of a simply multiplication table

Exercise 1. Consider the set **S** of finite non-empty strings of symbols from the alphabet *A*, as in Example 2.3. Can you think of other candidates for multiplication operations on **S**, besides the straightforward concatenation of strings considered above? Do your candidates define semigroup multiplications – that is, do they obey the associative law? For example, one might consider the operation where, given an ordered pair of strings, one first doubles the last symbol of the first string, and then concatenates. Is this operation associative?

See solution on page 31.

Example 2.4. The function $\max : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ defines a multiplication operation which equips $\mathbb{N}$ with the structure of a semigroup.

Exercise 2. Verify the statement made in Example 2.4; that is, check that the associative law holds.

Does min : $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ also define a semigroup structure on $\mathbb{N}$?

See solution on page 31.

Example 2.5. Consider the set $X = \{\text{sprout}, \text{young}, \text{mature}, \text{old}, \text{dead}\}$ which describes five possible states of a plant. Let $T : X \to X$ be the function that describes "development":

$$\begin{aligned} T(\text{sprout}) &= \text{young} \\ T(\text{young}) &= \text{mature} \\ T(\text{mature}) &= \text{old} \\ T(\text{old}) &= \text{dead} \\ T(\text{dead}) &= \text{dead} \end{aligned}$$

In other words, we think of T as the change of state of the plant during a given time interval (say, three months). Composing the function T with itself corresponds to considering multiples of the given time interval. For example, the function

$$T \fatsemi T \fatsemi T : X \to X$$

models the change over the course of nine months. In general, for the n-fold composition of T with itself we write T^n. The set $\mathbf{S} = \{T^n \mid n \in \mathbb{N}\}$ is a semigroup, with the multiplication given by the composition operation.

Example 2.6.

Definition 2.7 (Discrete-time linear systems)**.** A discrete-time linear time-invariant proper open system is defined by three matrices A, B, C. Together they give a recurrence of the type

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k \end{aligned} \tag{2.3}$$

If x has dimension $n \geq 1$, u dimension $m \geq 1$ and y dimension $p \geq 1$, then A has dimension $n \times n$, B has dimension $n \times m$, C has dimension $p \times n$.

Consider now the systems where $m = p$ (but possibly different n): these are systems with input and output of the same size. Hence we can compose them in series.

Give formulas for composition in series

Show that the composition is associative, and hence for a monoid

Induced n-ary multiplication

Given a semigroup $\langle S, \fatsemi \rangle$, for each $n \in \mathbb{N}$, we can define an induced n-ary multiplication operation

$$S^n \to S, \ \langle s_1, s_2, \ldots, s_n \rangle \mapsto s_1 \fatsemi s_2 \ldots \fatsemi s_n.$$

Rewrite the above in function definition extended form; give a name.

Thanks to the associative law, this is well-defined – that is, we do not need to set parentheses. We will say that an element $s \in S$ is an *n-fold multiplication*

```
# TODO
```

Figure 2.1.: `sm1.semigroup.yaml`

if it is in the image of this n-ary multiplication operation. At times we may not wish to specify the arity of the multiplication, in which case we just speak of a *multiplication*.

Code exercises

> *Remark.* In this course we assume, unles otherwise noted, that you have done the previous exercises, so that you can build on top of the code that you already have. The first exercise and tutorial was in Section 4.3.

Interface

```
class FiniteSemigroup(Semigroup, ABC):
  @abstractmethod
  def carrier(self) -> FiniteSet:
    ...
```

Representation

The format is shown in Fig. 2.1.

Exercise 1 (`TestFiniteSemigroupRepresentation`). Create a function to load the data.

```
class FiniteSemigroupRepresentation(ABC):
  @abstractmethod
  def load(self, s: str) -> FiniteSemigroup:
    """ Load the data  """

  @abstractmethod
  def save(self, m: FiniteSemigroup) -> str:
    """ Save the data  """
```

Constructors

Exercise 2. Given a set, construct the free semigroup.

```
@abstractmethod
def free(self, fs: FiniteSet) -> FiniteSemigroup:
  """ Construct the free semigroup on a set. """
```

2.2. Monoids [draft]

Algebraic structures are often defined in *layers*. For example, in the definition of semigroup, we start with a set **S** as a basic building block, and we add a

layer of structure to it, namely a multiplication operation $\fatsemi : \mathbf{S} \times \mathbf{S} \to \mathbf{M}$. The multiplication operation for semigroups was not only a new *structure* that we added, but we also required that this structure obey a *condition*, namely that it satisfy the associative law. One might also say that the multiplication operation was a new *constituent* or a new *datum*, and that satisfying the associative law is a *property*. Mathematicians often use such words in an intuitive, non-rigorous way as a tool for structuring their thinking. We will do the same. For clarity, we will aim to stick with the words *constituents* and *conditions*. Roughly speaking, we think of constituents as building blocks, and we think of conditions as rules for how those blocks fit together and behave.

Using the constituent vs. condition distinction we will, in particular, present some definitions in the following succinct, list-like fashion:

Definition 2.8 (Monoid)**.** A *monoid* $\mathbf{M}$ is:

Constituents

1. a set $\mathbf{M}$;
2. a binary operation $\fatsemi : \mathbf{M} \times \mathbf{M} \to \mathbf{M}$, called *multiplication*;
3. a specified element $\mathrm{id} \in \mathbf{M}$, called *neutral element*.

Conditions

1. Associative law: $(x \fatsemi y) \fatsemi z = x \fatsemi (y \fatsemi z)$;
2. Neutrality Laws: $\mathrm{id} \fatsemi x = x = x \fatsemi \mathrm{id}$.

Remark. The way that we presented the definition of a monoid is certainly not unique. For example, we could have done the following.

A *monoid* is:

Constituents

1. a semigroup $\langle \mathbf{M}, \fatsemi_{\mathbf{M}} \rangle$;
2. a specified element $\mathrm{id} \in \mathbf{M}$, called *neutral element*.

Conditions

1. Neutrality Laws: $\mathrm{id} \fatsemi_{\mathbf{M}} x = x = x \fatsemi_{\mathbf{M}} \mathrm{id}$.

In this version, two constituents and one condition from Definition 2.8 are "compressed" into the information that we are using here a semigroup as a constituent. This kind of "compression" has its pros and cons; depending on the context will use it to varing degrees.

There is a similar dilemma when considering the software interfaces to describe these structure. In terms of software engineering, the two strategies are *composition* (a monoid has a semigroup as a constituent) and *inheritance* (a monoid *is* a semigroup with additional data).

Example 2.9. Consider $\langle \mathbb{R}, +, 0 \rangle$. This is a monoid, since, for all $x, y, z \in \mathbb{R}$, we have

$$(x + y) + z = x + (y + z),$$

and

$$x + 0 = x = 0 + x.$$

Example 2.10. The set $\mathbb{Z}$, together with the operation of multiplication of whole numbers, forms a monoid. The neutral element is the number 1.

Lemma 2.11. Let $\langle \mathbf{S}, \fatsemi \rangle$ be a semigroup. If there exists elements $1 \in M$ and $1' \in M$ such that $\langle \mathbf{S}, \fatsemi, 1 \rangle$ and $\langle \mathbf{S}, \fatsemi, 1' \rangle$ are each monoids, then $1 = 1'$ must hold. In other words, the neutral element of a monoid is uniquely determined by the underlying semigroup structure.

Exercise 3. Prove Lemma 2.11.

See solution on page 31.

JL: I would change the example below; it feels slightly misleading to use "0". This construction works for any closed ray open toward $+\infty$

Example 2.12. Consider $\langle \mathbb{R}_{\geq 0}, \max, 0 \rangle$. This is a monoid, since, for all $x, y \in \mathbb{R}_{\geq 0}$, we have:

$$\max(\max(x, y), z) = \max(x, \max(y, z)),$$

and

$$\max(x, 0) = x = \max(0, x).$$

JL: We should edit / change the example below (for coherence with the string example for semigroups, and to change the definition of sequence)

Example 2.13 (Sequences)**.** A sequence is a function whose domain is a subset of $\mathbb{N}$ JL: this is a strange and non-standard definition of sequence; I would avoid, and are called *finite* if the domain of the function is finite. Often finite sequences are referred to as *lists*. Given a set S, we denote the set of all lists on S by S^*. This can be made into a monoid, by considering *concatenation* as the operation, and the empty list as the neutral element. Specifically, a list is an element $s \in S^*$ consists of a $n \in \mathbb{N}$ and a function $f : [n] \to S$, where $[n] = \{i : \mathbb{N} \mid i < n\} \subseteq \mathbb{N}$. The empty list, denoted (), is the unique list of length 0. Given $n > 0$, we can write the list which assigns $0, \dots, n-1$ to $s_0, s_1, \dots, s_{n-1}$ as $(s_0, s_1, \dots, s_{n-1})$. Given a list $x \in S^*$ of length m and a list $y \in S^*$ of length n, we can define their *concatenation* $x * y$ as list of length $m + n$ with:

$$i \mapsto \begin{cases} x_i & \text{if } i < m \\ y_{i-m} & \text{if } i \geq m. \end{cases}$$

Clearly, this definition of concatenation satisfies associativity and unitality, making this construction a monoid. This is often referred to as the *free monoid on* S.

Example 2.14 (Transition functions)**.**

Definition 2.15 (Continuous-time dynamical system)**.** A dynamical system on $\mathbb{R}^n$ is defined by a function

$$f : \mathbb{R}^n \to \mathbb{R}^n. \tag{2.4}$$

A trajectory of a dynamical system is a function $x : \mathbb{R} \to \mathbb{R}^n$ such that

$$\dot{x}_t = f(x_t). \tag{2.5}$$

We use the notation $\dot{x}$ to abbreviate $\mathrm{d}x/\mathrm{d}t$.

Suppose that we are dealing with dynamical systems such that for any point x_0, there is exactly one trajectory beginning at x_0. (For this we need to put reasonable constraints on the function f.)

We can then ask the following: given a point x, where would its trajectory be after δ? This question induces a family of functions T_δ, called transition functions. For each particular δ, we have a function

$$T_\delta : \mathbb{R}^n \to \mathbb{R}^n \tag{2.6}$$

that maps a point to its position δ in the future.

We can spot here a semigroup structure. Suppose we want to know the position of a point $\delta_1 + \delta_2$ in the future. We can take T_{δ_1} and compose it with T_{δ_2}; or take directly $T_{\delta_1+\delta_2}$. By construction we will have that

$$T_{\delta_1+\delta_2} = T_{\delta_1} \fatsemi T_{\delta_2} \tag{2.7}$$

We can also easily prove associativity. This shows that the set of transition functions for a particular system with the operation of function composition form a semigroup.

This semigroup is a monoid because there is an identity. The identity is T_0, the map that tells us what happens after 0 seconds. That is $\mathrm{id}_{\mathbb{R}^n}$, the identity on $\mathbb{R}^n$. To show that $T_0 = \mathrm{id}_{\mathbb{R}^n}$ is an identity, we can fix any δ and substituting in (2.7) we have

$$\begin{aligned} & T_{\delta+0} \\ =& T_\delta \fatsemi T_0 \\ =& T_\delta \end{aligned} \tag{2.8}$$

Exercise 4. [Discrete-time linear systems] In Example 2.6 we have constructed the semigroup of linear discrete-time dynamical systems.

1. Show that it is not a monoid, by showing that one cannot find an identity.
2. Suggest an extension to **??** so that you can obtain a monoid.

See solution on page 31.

Code exercises

Interface

```
class Monoid(Semigroup, ABC):
  @abstractmethod
  def identity(self) -> Element:
    ...
```

```
class FiniteMonoid(Monoid, FiniteSemigroup, ABC):
  """"""
```

Representation

The format is shown in Fig. 2.2.

```
carrier:
  elements: [ a, b, c, e ]
identity: e
```

Figure 2.2.: `monoid1.monoid.yaml`

Exercise 3 (Representation). Create a function to load the data.

```
class FiniteMonoidRepresentation(ABC):
  @abstractmethod
  def load(self, s: str) -> FiniteMonoid:
    """ Load the data  """

  @abstractmethod
  def save(self, m: FiniteMonoid) -> str:
    """ Save the data  """
```

2.3. Rope Goldberg Machines

The inventions of Professor Butts, transcribed by Rube Goldberg, are prodigious machines that provide practical ways to perform everyday tasks (Fig. 2.3). They also highlight the power of compositionality.

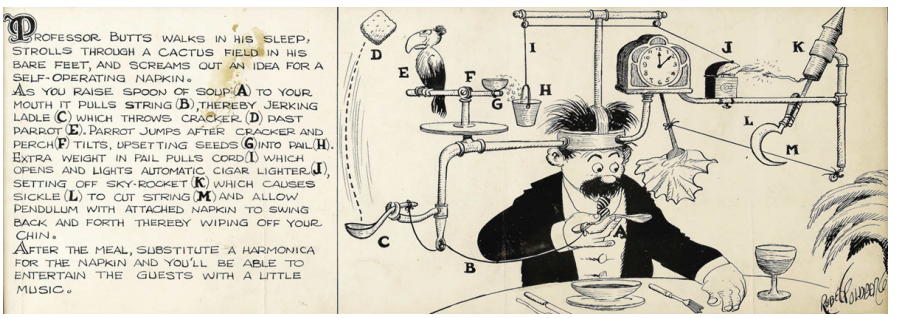
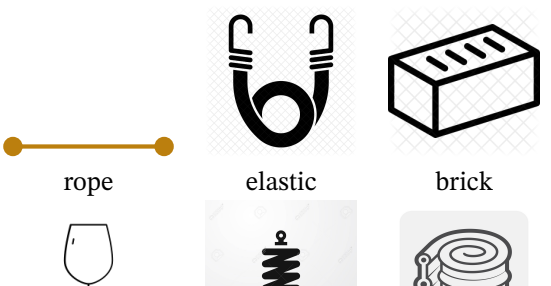


Figure 2.3.: *Life's Little Jokes #59,380*, Rube Goldberg

We are going to assemble similar machines. We start by introducing some of the essential elements:

- Rope
- Elastic
- Brick
- Glass
- Spring
- Rubber

work on original icons



These different elements can be classified by their elasticity properties, and hence by the way in which they deform. A good reference book for these concepts is Hibbeler [**hibbeler2014mechanics**]. Given any material, it reacs to the application of specific forces. The force applied per unit area is referred to as *stress*, and the stretching/compression produced as a response from the material is called *strain*. The strain ε is usually written as the ratio of the difference in length along the direction of the stress ΔL, and the original length of the material L_0:

$$\varepsilon = \frac{\Delta L}{L_0}$$

When stress is applied, each material experiences a specific strain, which depends on the chemical bonds creating the material. Whether the material returns to its original shape when stress is removed, depends on the entity of deformation. One usually differentiates between *elastic* deformation (when removing the stress makes the material return to its original shape), and *plastic* deformation (when the material deforms irreversibly): see Fig. 2.4. If one observes the stress vs. strain curve, the first region behaves linearly, meaning that the force required to deform an elastic material is directly proportional to its deformation. This is commonly known as the Hooke's law, which is due to the physicist Robert Hooke:

$$F = -k\Delta x,$$

where F represents the force, Δx the deformation of the material, and k the so-called *spring constant* (in [N/m]).

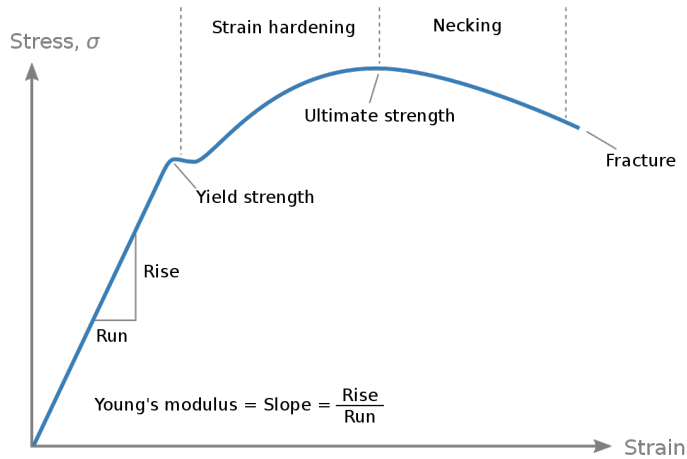


Figure 2.4.:

do our one

.

Example 2.16. You are skiing in the beautiful Swiss alps. Your friend weighs 100 kg and sits on a chairlift with spring constant 4,000 N/m. The nominal extension of the chairlift is 0.25 m. The weight of your friend will create an extension of the chairlift, which can be computed as:

$$\begin{aligned}\Delta x &= \frac{F}{k} \\ &= \frac{mg}{k} \\ &\approx 0.15\,\text{m}.\end{aligned}$$

One can therefore compute the extension as $L = L_0 - \Delta x = 0.1\,\text{m}$.

The ability of a material to being deformed elastically is described by the so-called Young's *modulus*, name of which is due to the physicist Thomas Young. This modulus, usually named E is defined as

$$E = \frac{\text{stress}}{\text{strain}}.$$

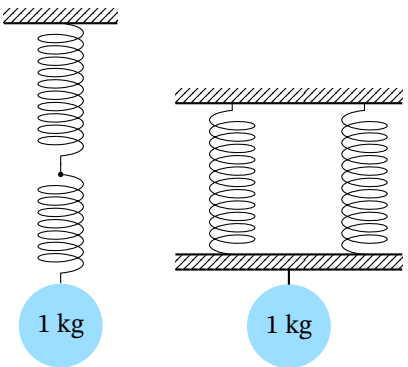
Note that E is a constant if computed in the domain of validity of Hooke's law. In this domain, we can obtain the spring constant k by using the formula:

$$k = E\frac{A}{L_0},$$

where L_0 is the nominal length of the material, and A is the area over which the force is applied.

It is interesting to see what happens when one combines multiple materials. The first two elastic material combinations which come to mind are *series* and *parallel*. If you take two materials with the same spring constant k, you concatenate them, and you hang a weight at their lower end (**??**), they will be equivalent to one single spring with double the length. The effective spring constant of the combination must therefore be halved to $k/2$. In case one puts the two materials in parallel (**??**), the length remains the same, and the resulting spring constant doubles to $2k$.

This is the exact same reasoning for capacitors in electrical circuits (isomorphic example for ITET people:))



Describe here the properties of these components (elasticity, etc.)

These are the classes that define the data.

```
class Component:
  mass: float
```

```
class Rope(Component):
  length: float
```

```
class Brick(Component):
  pass
```

```
class Elastic(Rope):
  elasticity: float  # meaning?
```

Exercise 4 (`TestRopeGoldberg`). In this exercise you will compute the behavior of a chain of those components.

The interface you need to implement is:

(a) Hanging the chain

(b) Horizontal compression

(c) Horizontal extension

Figure 2.5.: Visualization of the three scenarios.

Make graphics.

```python
class RupeGoldbergSolver(ABC):

  def hangthem(self, components: List[Component]) -> Solution:
    """ What if we hang the first component, and let the others hang below it?  """

  def push(self, components: List[Component], force: float) -> Solution:
    """ What if we fix one endpoint, and we compress it? """

  def pull(self, components: List[Component], force: float) -> Solution:
    """ What if we fix one endpoint, and we pull the other end? """
```

1. The first function asks what would happen if you were to hang this chain on one end, and let it hang (Fig. 2.5a). Would it break? If not, what would be the extended length? GZ: The "'would it break" question is not as smooth as it seems. The strain-stress diagrams have nonlinearities and maybe we end up in oversimplifying.
2. The second question asks what would happen if you were to let the chain rest horizontally (Fig. 2.5b). Then fix one end to an *immovable wall* and push to the other end. What would happen? If the push destroys some of the objects, consider them destroyed and of length 0.
3. The third question asks what would happen if you were to pull with a certain force (Fig. 2.5c). Either you will break the chain, or there will be an equilibrium. What would be the length?

We will walk you through the theoretical part, but we will leave the implementation to you.

We should think *compositionally*. We should first see what happens in the specific cases, and see if the answer can be generalized in a way that the solution is compositional.

Go through the specific cases

Give breakthrough observation that all the behavior can be described by one relation between force and length. Hence what we are talking about is really composition in a certain monoid.

The solution is to create for each component this relation, and then compose the relations.

2.4. Generators and relations [draft]

In Example 2.5 we considered a set of states $X = \{\text{sprout, young, mature, old, dead}\}$, a function $T : X \to X$, and the semigroup

$$\mathbf{S} = \{T^n \mid n \in \mathbb{N}\}. \tag{2.9}$$

Note that $\mathbf{S}$ has a special form: all of its elements can be expressed in terms one of its elements, T, and the multiplication operation (which, in this case, is function composition). To describe this state of affairs we say that $\mathbf{S}$ is *generated* by the element T.

Recall that T was defined by

$$\begin{aligned} T(\text{sprout}) &= \text{young} \\ T(\text{young}) &= \text{mature} \\ T(\text{mature}) &= \text{old} \\ T(\text{old}) &= \text{dead} \\ T(\text{dead}) &= \text{dead}. \end{aligned}$$

Observe that the function T^4 will map all elements of X to the element "dead". For example, if we start with the element "sprout", the result of applying T four times is

$$\text{sprout} \overset{T}{\to} \text{young} \overset{T}{\to} \text{mature} \overset{T}{\to} \text{old} \overset{T}{\to} \text{dead}.$$

Note also that for *any* $n \geq 4$, the function T^n will map all elements of X to the element "dead". If we consider T^6, for example, then, for any $x \in X$,

$$T^6(x) = T^2(T^4(x)) = T^2(\text{dead}) = T(T(\text{dead})) = T(\text{dead}) = \text{dead}.$$

It follows that all T^n, for $n \geq 4$, are actually *all the same map*: the one that sends every state to the dead state. Thus $\mathbf{S} = \{T^n \mid n \in \mathbb{N}\}$ actually only has at most four elements! Namely T, T^2, T^3, and T^4. (Are any of these four maps actually equal, too?)

When two elements which a priori could be distinct from each other (such as T^6 and T^4 above) turn out to be equal, we call this a *relation* between the elements of $\mathbf{S}$.

Definition 2.17 (Generated semigroups)**.** Let $\langle \mathbf{S}, \fatsemi \rangle$ be a semigroup, and let $\mathbf{A} \subseteq \mathbf{S}$ be a subset. We say that $\mathbf{S}$ is *generated* by $\mathbf{A}$ if every element of $\mathbf{S}$ can be expressed as a finite multiplication of elements of $\mathbf{A}$.

Example 2.18. Consider Example 2.3, where elements of the semigroup $\mathbf{S}$ were non-empty strings built using the elements of the "alphabet" set $\mathbf{A} = \{\bullet, \bullet\}$. In this case, $\mathbf{S}$ is generated by $\mathbf{A}$.

Example 2.19. Consider the natural numbers (without zero) as a semigroup, where addition is the semigroup multiplication (see Example 2.2). This semigroup is generated by the subset $\{1\}$.

Definition 2.20. A *relation* on a semigroup $\langle \mathbf{S}, \fatsemi \rangle$ is an equation between two multiplications of elements of $\mathbf{S}$.

Example 2.21. Consider the semigroup $\langle \mathbb{N}, + \rangle$, and consider $l, k \in \mathbb{N}$. The equation $l + k = k + l$ is an example of a relation.

say that similar notions/defintions also apply for monoids

introduce free semigroups and free monoids

introduce and explain presentations of semigroups and monoids via generators and relations

2.5. Groups [draft]

Definition 2.22 (Group)**.** A *group* is a monoid together with an "inverse" operation. In more detail, a group is

Constituents

1. a set $\mathbf{M}$;
2. a binary operation $\fatsemi : \mathbf{M} \times \mathbf{M} \to \mathbf{M}$, called *composition*;
3. a specified element $\mathrm{id} \in \mathbf{M}$, called *neutral element*.
4. a map $\mathrm{inv} : \mathbf{M} \to \mathbf{M}$ called "inverse".

Conditions

1. Associative law: $(x \fatsemi y) \fatsemi z = x \fatsemi (y \fatsemi z)$;
2. Neutrality Laws: $\mathrm{id} \fatsemi x = x = x \fatsemi \mathrm{id}$.
3. Inverse law:

$$\mathrm{inv}(x) \fatsemi x = \mathrm{id} = \mathrm{inv}(x) \fatsemi x \tag{2.10}$$

Example: square matrices with full rank

Example-corollary: rototranslations SE(2), SE(3)

3. Homomorphisms and actions [draft]

In this chapter we look at homomorphisms, which are maps between two semigroups (or monoids, groups) that "preserve the structure". We also look at *actions*, which describe some form of compositional effects of a semigroup into another set.

3.1 Homomorphisms 25
3.2 Monoid homomorphism [missing] 28
3.3 Group homomorphism 29
3.4 Actions [missing] 29

3.1. Homomorphisms

Definition 3.1 (Semigroup homomorphism)**.** Let $\mathbf{S}$ and $\mathbf{T}$ be semigroups. A homomorphism of semigroups from $\mathbf{S}$ to $\mathbf{T}$ is a function $F : \mathbf{S} \to \mathbf{T}$ such that for all $x, y \in \mathbf{S}$,

$$F(x \mathbin{\circ_{\mathbf{S}}} y) = F(x) \mathbin{\circ_{\mathbf{T}}} F(y) \tag{3.1}$$

We think of (3.1) as a way of saying that the function of sets $F : \mathbf{S} \to \mathbf{T}$ is *compatible* with the multiplication operations on $\mathbf{S}$ and $\mathbf{T}$, respectively.

Definition 3.2 (Identity homomorphism)**.** Let $\mathbf{S}$ be a semigroup. The identity function $\mathrm{Id}_{\mathbf{S}} : \mathbf{S} \to \mathbf{S}$ is always a morphism of semigroups. Indeed, the

condition

$$\mathrm{Id}(s_1 \fatsemi_{\mathbf{S}} s_2) = \mathrm{Id}(s_1) \fatsemi_{\mathbf{S}} \mathrm{Id}(s_2) \quad \forall s_1, s_2 \in \mathbf{S} \tag{3.2}$$

is satisfied. We call this the *identity homomorphism* of **S**.

Definition 3.3 (Semigroup isomorphism)**.** Let **S** and **T** be semigroups. A homomorphism of semigroups $F : \mathbf{S} \to \mathbf{T}$ is called a *semigroup isomorphism* if there exists a homomorphism of semigroups $G : \mathbf{T} \to \mathbf{S}$ such that

$$F \fatsemi G = \mathrm{Id}_{\mathbf{S}} \text{ and } G \fatsemi F = \mathrm{Id}_{\mathbf{T}}. \tag{3.3}$$

Example 3.4 (Logarithms)**.** The positive reals with multiplication $\langle \mathbb{R}_{>0}, \cdot \rangle$ is a semigroup. The reals with addition $\langle \mathbb{R}, + \rangle$ is a semigroup.

Now consider as a bridge between the two the logarithmic function

$$\log : \mathbb{R}_{>0} \to \mathbb{R} \tag{3.4}$$

and its inverse

$$\exp : \mathbb{R} \to \mathbb{R}_{>0} \tag{3.5}$$

We already know that these are inverse of each other:

$$\begin{aligned} \exp \fatsemi \log &= \mathrm{id}_{\mathbb{R}} \\ \log \fatsemi \exp &= \mathrm{id}_{\mathbb{R}_{>0}} \end{aligned} \tag{3.6}$$

We can verify that log is also a semigroup homomorphism, because of this property of the logarithms:

$$\log(a \cdot b) = \log(a) + \log(b). \tag{3.7}$$

Because log is a bijection and exp is its inverse, it already follows that exp is a homomorphism in the opposite direction. Alternatively we can see that is the case because of the property of the exponentials:

$$\exp(c + d) = \exp(c) \cdot \exp(d). \tag{3.8}$$

Equations (3.7) and (3.8) are both (3.1) in disguise.

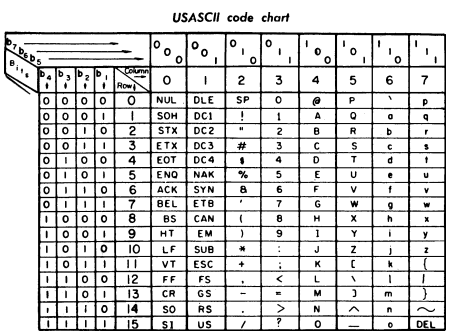


Figure 3.1.: 7-bit US-ASCII encoding

Example 3.5 (ASCII code)**.** ASCII encoding takes any alphanumerical characters and symbols into a number between 0 and 127 (Fig. 3.1). Let's call $\mathtt{A}$ the set of those 127 symbols. We can see ASCII encoding as a semigroup homorphism of $\mathtt{A}^*$ to the free semigroup on the integers $[0, 127]^*$.

Because we can also go back, by finding the inverse function, ASCII encoding is also an isomorphisms of semigroups.

Example 3.6 (ASCII code to binary)**.** Currently, computers use binary to store data. (There were, in fact, *trinary* computers.) In Fig. 3.1, you can see represented also the binary encoding of each character. Therefore, we can see ASCII as a homomorphism between $\mathtt{A}^*$ and binary strings $\{0, 1\}^*$.

Exercise 5. Show that the homomorphism

$$\text{ASCII} : \mathtt{A}^* \to \{0,1\}^* \tag{3.9}$$

is *not* an isomorphism.

See solution on page 31.

Example 3.7 (Morse code)**.** Consider the Morse code: a way to encode the letters and numerals to an alphabet of dots (•) and dashes (—). The encoding is shown in Table 3.1. Here, the alphabet $\mathcal{A}$ is the letters A–Z and the numbers 0–9. There is no difference between upper and lower case, and there are no punctuation marks.

Transcribing a text in Morse code is not just a matter of creating the right sequence of dots and dashes. The standard also requires a certain timing of the events. If the length of • is 1, then the length of — must be 3. There must be an interval of 3 between letters, and 7 between words.

Therefore, there are 5 symbols in the Morse alphabet (Table 3.2); each representing a *signal*.

Define now the extended alphabet $\mathcal{A}'$ to be the union of $\mathcal{A}$ and the set {▮, ▮}, where ▮ is interletter space, and ▮ is inter-word space.

Therefore, to encode the sentence

$$\text{“I am well”} \tag{3.10}$$

we first transform it to upper case

$$\text{“I AM WELL”} \tag{3.11}$$

Then we note the inter-letter space and the interword spaces:

$$\mathtt{I}\,▮\,\mathtt{A}\,▮\,\mathtt{M}\,▮\,\mathtt{W}\,▮\,\mathtt{E}\,▮\,\mathtt{L}\,▮\,\mathtt{L} \tag{3.12}$$

At this point we can substitute the Morse code to obtain

$$\bullet\, s_1 \bullet s_7 \bullet s_1 - s_3 - s_1 - s_7 \bullet s_1 - s_1 - s_3 \bullet s_3 - s_3 - \tag{3.13}$$

In signal space—what somebody would hear—this becomes

(3.14)

With this representation it is clear that 5 symbols are redundant: if we have a 1-period beep and a 1-period silence, we can obtain the 3-period silence and beeps and the 7-period silence.

In the end, the Morse alphabet is *binary* in the sense that it all reduces to two symbols: not {•, —} but rather the alphabet {▮, ▮}.

Exercise 6. We have seen that Morse code transforms a word in the alphabet

$$\mathcal{A} = (\mathtt{A} \text{ to } \mathtt{Z}) \cup (0 \text{ to } 9) \cup \{▮\} \tag{3.15}$$

to the alphabet

$$\mathcal{B} = \{▮, ▮\} \tag{3.16}$$

Table 3.1.: Morse encoding

A	•—		
B	—•••		
C	—•—•		
D	—••		
E	•		
F	••—•		
G	——•		
H	••••		
I	••	**0**	—————
J	•———	**1**	•————
K	—•—	**2**	••———
L	•—••	**3**	•••——
M	——	**4**	••••—
N	—•	**5**	•••••
O	———	**6**	—••••
P	•——•	**7**	——•••
Q	——•—	**8**	———••
R	•—•	**9**	————•
S	•••		
T	—		
U	••—		
V	•••—		
W	•——		
X	—••—		
Y	—•——		
Z	——••		

Table 3.2.: 5 symbols for Morse encoding

•	beep of length ℓ	▮
—	beep of length 3ℓ	▮
s_1	silence of length ℓ	▮
s_3	silence of length 3ℓ	▮
s_7	silence of length 7ℓ	▮

Is this map a homomorphism of semigroups?

See solution on page 32.

Example 3.8 (Phonetic languages)**.**

Nice example: the map from sequence of characters to sequences of sounds is monoidal in certain languages (Korean, Japanese, almost in Italian.) and also invertible.

Example 3.9 (State dimension of discrete dynamical systems)**.**

Example of the map $f : \mathrm{DDS} \to \mathbb{N}$ that gives the size of the state.

Example 3.10 (Transition function, continuation of Example 2.14)**.** Consider the map $f : \mathbb{R}_{\geq 0} \to (\mathbb{R}^n \to \mathbb{R}^n)$ that associates to a delta δ its transition function T_δ. Re-reading (2.7), we can see that it is a homomorphism between the semigroup $\langle \mathbb{R}_{\geq 0}, + \rangle$ and the semigroup of endormorphisms of $\mathbb{R}^n$.

Exercise 7. How many different non-isomorphic semigroups are there with precisely one element? How many with precisely two elements?

See solution on page 32.

Exercise 8. Let $F : \mathbf{S} \to \mathbf{T}$ be a morphism of semigroups. Prove that F is an isomorphism of semigroups if and only if the function $F : \mathbf{S} \to \mathbf{T}$ is bijective.

See solution on page 32.

3.2. Monoid homomorphism [missing]

Definition 3.11 (Monoid homomorphism)**.** Let $\langle \mathbf{M}, \fatsemi_{\mathbf{M}}, 1_{\mathbf{M}} \rangle$ and $\langle \mathbf{N}, \fatsemi_{\mathbf{N}}, 1_{\mathbf{N}} \rangle$ be monoids. A homomorphism of monoids from $\mathbf{M}$ to $\mathbf{N}$ is a function $f : \mathbf{M} \to \mathbf{N}$ such that

$$f(m_1 \fatsemi_{\mathbf{M}} n_2) = f(m_1) \fatsemi_{\mathbf{N}} f(n_2) \quad \forall m_1, n_2 \in \mathbf{M} \tag{3.17}$$

and

$$f(1_{\mathbf{M}}) = 1_{\mathbf{N}} \tag{3.18}$$

Example 3.12. The set $\mathbf{M} = \{-1, 0, 1\}$, together with multiplication of whole numbers and with 1 as neutral element, forms a monoid. The inclusion map $\mathbf{M} \to \mathbb{Z}$ is a homomorphism of monoids.

Nice example: the map from sequence of characters to sequences of sounds is monoidal in certain languages (Korean, Japanese, almost in Italian.) and also invertible.

Definition 3.13 (Identity homomorphism)**.** Let $\mathbf{M}$ be a monoid. Similar to the case of semigroups, the identity function $\mathrm{id}_{\mathbf{M}} : \mathbf{M} \to \mathbf{M}$ is also a homomorphisms of monoids.

Definition 3.14 (Monoid isomorphism)**.** A homomorphism of semigroups $f : \mathbf{M} \to \mathbf{N}$ is called a *monoid isomorphism* if there exists a homomorphism of monoids $g : \mathbf{N} \to \mathbf{M}$ such that

$$f \fatsemi g = \mathrm{id}_{\mathbf{M}} \qquad \text{and} \qquad g \fatsemi f = \mathrm{id}_{\mathbf{N}}\,. \tag{3.19}$$

Exercise 9. Prove: a morphism of monoids $f : \mathbf{M} \to \mathbf{N}$ is an isomorphism of monoids if and only if the function $f : \mathbf{S} \to \mathbf{T}$ is bijective.

See solution on page 32.

3.3. Group homomorphism

Show the inverse operation being compatible with group structure, commuting with homomorphisms. This is the simplest example of a dagger category, to be explained later on.

3.4. Actions [missing]

Example 2.5 is an option for introducing the notion of action, or just as an example

Make example of dynamical systems acting on sequences.

Mention group actions and give some simple examples, e.g. in terms of symmetries

4. Solutions to selected exercises

Solution of 1.

Write solution of Exercise 1.

Solution of 2.

Write solution of Exercise 2.

Solution of 3.

to write solution

Solution of 4.

to complete

AC: The extension needs to introduce the matrix D to have a direct i-o signal. ALSO, we need to change the formalization to allow $n = 0$, otherwise you cannot make an identity because the state space grows by at least 1 when you compose with it.

Solution of 5. We can show that we cannot find an inverse morphism

$$g : \{0,1\}^* \to \mathtt{A}^* \tag{4.1}$$

At first sight everything seems in order: if we could find an isomorphism into $[0,127]^*$ and we can express integers in binary, what could hold us back?

What fails here is something so simple it could go unnoticed: the hypothetical function g is not well defined for all points of its domain. We know how to

translate a binary string of length $7, 14, 21, \ldots$ back to symbols; but what would be the output of g on the string 111?

The function g is a left inverse for ASCII, in the sense that $\text{ASCII} \fatsemi g = \text{id}_{\mathtt{A}^*}$, but it is not a right inverse.

Solution of 6. The answer is **no** because the encoding is context dependent; I don't know if a single letter is followed by a space or another letter. For example, take the string

$$I\,AM\,MAX \tag{4.2}$$

We can decompose it as follows

$$I\,A \fatsemi M \fatsemi \,\fatsemi M \fatsemi AX \tag{4.3}$$

If Morse encoding was a homomorphism F then we would be able to encode the string as follows:

$$F(I\,A) \fatsemi F(M) \fatsemi F(\,) \fatsemi F(M) \fatsemi F(AX) \tag{4.4}$$

However, this cannot work, because in the second instance of M we would need to output a letter separator, while in the first case we don't.

Can you find a way to fix it?

For example you can consider the alphabet

$$((\mathtt{A} \text{ to } \mathtt{Z}) \cup (0 \text{ to } 9)) \times \{\,,\,\} \tag{4.5}$$

where we annotate if each symbol is followed by a letter or by a space.

In this representation, the string can be written as

$$\langle I,\,\rangle\langle A,\,\rangle\langle M,\,\rangle\langle M,\,\rangle\langle A,\,\rangle\langle X,\,\rangle\,. \tag{4.6}$$

Based on this representation we can define context-independent rules that make a homomorphism.

Solution of 7.

Write solution of Exercise 7.

Solution of 8.

Write solution of Exercise 8.

Solution of 9.

Write solution