Nicola.Botta@pik-potsdam.de

Week 7 DSLofMath

- Understand basic notions of LA from a CS angle
- Apply these notions ...
- Generalize HV computations to monoids

- Skills:
  - read Chapter 7 of DSLofMath book
  - solve exercises

Chapter 7

Elements of Linear Algebra

Often, especially in engineering textbooks, one encounters the following definition: a vector is an $n+1$-tuple of real or complex numbers, arranged as a column:

$$v = \begin{bmatrix} v_0 \\ \vdots \\ v_n \end{bmatrix}$$

Other times, this is supplemented by the definition of a row vector:

$$v = \begin{bmatrix} v_0 & \cdots & v_n \end{bmatrix}$$

The v_is are real or complex numbers, or, more generally, elements of a *field* (See Section 4.1.1 for the definition of a field).

Vectors and their spaces However, following our theme, we will first characterise vectors algebraically. From this perspective a *vector space* is an algebraic structure that captures a set of vectors, with zero, a commutative addition, and scaling by a set of scalars (i.e., elements of the field). In terms of type classes, we can characterise this structure as follows:

```
class (Field s, AddGroup v) ⇒ VectorSpace v s where
  (◁) :: s → v → v
```

The class declaration is short, but can need some unpacking. First, the type s for the scalars, is required to be a field, which basically means we have $(+)$, $(-)$, $(*)$, and $(/)$ available as operations on values of type s, the scale

Nicola.Botta@pik-potsdam.de

recap 4.1.1 :

Field = Ring ∧ MulGroup

Ring = AddGroup + Mul

MulGroup = MUL + (/, rec)

Mul = (*, one)

AddGroup = Add + (-, neg)

Add = (+, zero)

VectorSpace V S =
Field S ∧ AddGroup V ∧
(◁) ∧ ...

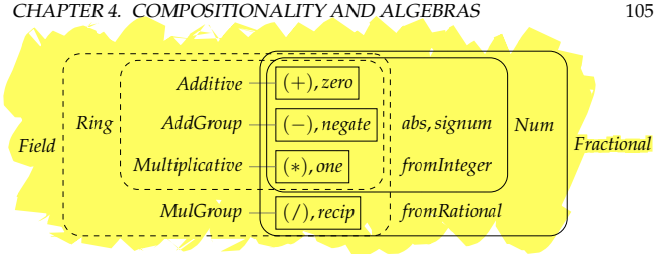


Figure 4.1: Comparing the Haskell Prelude class hierarchy (*Num*, *Fractional*) with the book's hierarchy. In addition to the groupings visible in the figure, the class *AddGroup* includes the *Additive* operations and *MulGroup* includes the *Multiplicative* operations.

4.2 Homomorphisms

The Wikipedia definition of homomorphism (2021-12-27) states that "A homomorphism is a structure-preserving map between two algebraic structures of the same type". We will spend the next few subsections on formal definitions and examples of homomorphisms.

4.2.1 (Homo)morphism on one operation

As a stepping stone to capture the idea of homomorphism, we can define a ternary predicate H_2. The first argument h, is the map. The second (Op) and third (op) arguments correspond to the algebraic structures.

$$H_2(h, Op, op) = \forall\, x.\ \forall\, y.\ h\ (Op\ x\ y) == op\ (h\ x)\ (h\ y)$$

If the predicate $H_2(h, Op, op)$ holds, we say that $h : A \to B$ is a homomorphism from $Op : A \to A \to A$ to $op : B \to B \to B$. Or that h is a homomorphism from Op to op. Or even that h is a homomorphism from A to B if the operators are clear from the context. We have seen several examples in earlier chapters:

1. in Section 1.4 we saw that $evalE : ComplexE \to ComplexD$ is a homomorphism from the syntactic operator *Plus* to the corresponding semantic operator *plusD*.
2. in Chapter 2 we saw De Morgan's laws, which say that "not" ($\neg$) is a homomorphism in two ways: $H_2(\neg, (\wedge), (\vee))$ and $H_2(\neg, (\vee), (\wedge))$.

factors. Then, the type v needs to be an additive group, thus supporting a zero vector, $(+)$ and $(-)$ on vectors. These operations are all required before we are allowed to declare a *VectorSpace* instance, due to the constraint $(Field\ s, AddGroup\ v)$. Finally, the new operator $(\triangleleft)$, called "scale" or scalar-vector-multiplication, takes a scale factor and a vector to a suitably resized vector: $2 \triangleleft v$ is twice v, while $(-1) \triangleleft v$ has the same length as v but points in the opposite direction, etc.

Laws Additionally, every vector space must satisfy the following laws:

1. Vector scaling $((s\triangleleft) :: v \to v)$ is a homomorphism over (from and to) the additive group structure of v. Thus for all vectors a and b we have:

$$\begin{aligned} s \triangleleft (a + b) &= s \triangleleft a + s \triangleleft b \\ s \triangleleft zero &= zero \\ s \triangleleft (negate\ a) &= negate\ (s \triangleleft a) \end{aligned}$$

   This means that scaling can be "pushed inside" any sum or difference.

2. On the other side, $(\triangleleft a)$ is a homomorphism from the additive group structure of s to the group structure of v. Thus, for all scalars s and t we have:

$$\begin{aligned} (s + t) \triangleleft a &= s \triangleleft a + t \triangleleft a \\ zero \triangleleft a &= zero \\ negate\ s \triangleleft a &= negate\ (s \triangleleft a) \end{aligned}$$

   For the examples above this means that $2 \triangleleft v == (1 + 1) \triangleleft v == 1 \triangleleft v + 1 \triangleleft v$ and $(-1) \triangleleft v == negate\ (1 \triangleleft v)$.

3. Finally $(\triangleleft)$ is a homomorphism from the multiplicative monoid of s to the monoid of endofunctions over v (see Section 4.1). Thus, for all scalars s and t we have:

$$\begin{aligned} (\triangleleft)\ one &= id \\ (\triangleleft)\ (s * t) &= (\triangleleft)\ s \circ (\triangleleft)\ t \end{aligned}$$

   Applying the functions to the vector a everywhere gives the familiar form for these laws:

$$\begin{aligned} one \triangleleft a &= id\ a \\ (s * t) \triangleleft a &= ((s\triangleleft) \circ (t\triangleleft))\ a \end{aligned} \qquad \begin{aligned} a &= a \\ &= s \triangleleft (t \triangleleft a) \end{aligned}$$

   For the examples above this means that $2 \triangleleft v == 1 \triangleleft v + 1 \triangleleft v == v + v$ (or "twice v") and $(-1) \triangleleft v == negate\ (1 \triangleleft v) == negate\ v$ (or "v in the opposite direction").

Nicola.Botta@pik-potsdam.de

$(s\triangleleft) : V \to V$

$(+) : ?$

$(\triangleleft v) : ?$

$(+) : ?$

$(\triangleleft) : ?$

Exercise: formulate in terms of H_0, H_1, H_2 !

Often, the above laws are not expressed in terms of homomorphisms, but rather as individual equations. This means that some of them are often omitted, because they are consequences of sets of other laws.

One-dimensional spaces We get the simplest instance declaration if we note that we can see scalars (like $\mathbb{R}$) as one-dimensional vectors with $s = v$:

$$\textbf{instance}\ \mathit{Field}\ s \Rightarrow \mathit{VectorSpace}\ s\ s\ \textbf{where}\ (\triangleleft) = (*)$$

Here (for once) the vectors have the same type as the scalars, which means that the scaling operation, which usually has an asymmetric type, now is just ordinary scalar multiplication $(*) :: s \to s \to s$. But for the rest of this chapter we will stick to the general case of n- (or infinite-) dimensional spaces.

Bases and representations An important consequence of the algebraic structure of vectors is that they can be expressed as a simple sort of combination of other special vectors. More precisely, we can *uniquely* represent any vector v in the space in terms of a fixed set of *basis* vectors $\{b_0, \ldots, b_n\}$. By definition, basis vectors cover the whole space:

$$\forall\, v.\ \exists\, s_0, \ldots, s_n.\ v == s_0 \triangleleft b_0 + \ldots + s_n \triangleleft b_n$$

They are also *linearly independent*:

$$(s_0 \triangleleft b_0 + \ldots + s_n \triangleleft b_n = 0) \Leftrightarrow (s_0 = \ldots = s_n = 0)$$

One can prove the uniqueness of representation as follows:

Proof. Assume two representations of v, given by s_i and t_i. The difference of those representations is given by $s_i - t_i$. But because they represent the same vector, their difference must be equal to the zero vector: $(s_0 - t_0) \triangleleft b_0 + \ldots + (s_n - t_n) \triangleleft b_n = 0$. By the basis being linearly independent, we find $s_i - t_i = 0$, and thus $s_i = t_i$. □

Syntax for vectors According to our red thread, this representation (coefficients) is akin to the notion of syntax. But this is a case where the representation is *equivalent* to the algebraic definition: the evaluator is not only a homomorphism, but an isomorphism between the space of vectors and the list of coefficients. This equivalence is what justifies the definition of vectors as columns (or rows) of numbers.

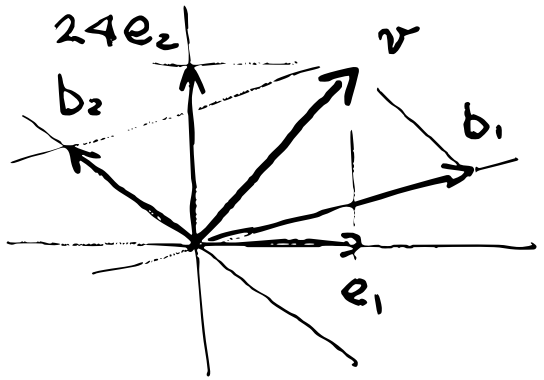
Nicola.Botta@pik-potsdam.de

Field S => VS S S

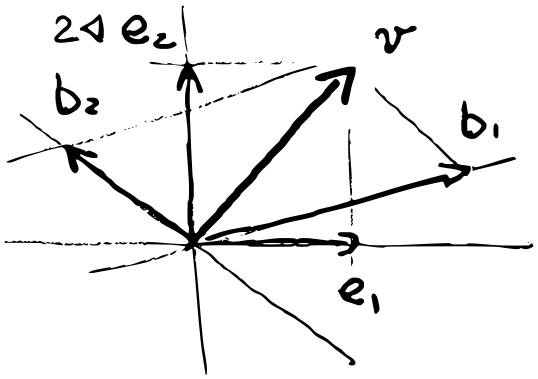
Exercise: show Hal S ...

2.1 s ◁ (p+q)
= { ? }
s ◁ p + s ◁ q

① Intuition is fine!



Nicola.Botta@pik-potsdam.de



$$v = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$b_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad b_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$$

$$v = 1 \triangleleft e_1 + 2 \triangleleft e_2$$

$$= 1 \triangleleft b_1 + 1 \triangleleft b_2$$

↑ coeff.

It's useful to remember that

- $b_0 \dots b_n$ linearly independent
- B s.t. $\mathrm{col}\, B = b_0 \dots b_n$ has rank $n+1$
- $B * s = v$ has unique solution

Nicola.Botta@pik-potsdam.de

Indeed, we can define:

$$v = \begin{bmatrix} v_0 \\ \vdots \\ v_n \end{bmatrix} = v_0 \triangleleft \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} + v_1 \triangleleft \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} + \cdots + v_n \triangleleft \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

So, for our column vectors, we can define the operations as follows:

$$v + w = \begin{bmatrix} v_0 \\ \vdots \\ v_n \end{bmatrix} + \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} v_0 + w_0 \\ \vdots \\ v_n + w_n \end{bmatrix}$$

$$s \triangleleft v = \begin{bmatrix} s * v_0 \\ \vdots \\ s * v_n \end{bmatrix}$$

In the following we denote by

$$e_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \leftarrow \text{position } k$$

the canonical basis vectors, i.e. e_k is the vector that is everywhere zero except at position k, where it is one, so that $v == v_0 \triangleleft e_0 + \ldots + v_n \triangleleft e_n$. This formula maps the syntax (coefficients, v_i) to the semantics (a vector, v).

Exercise 7.1. Define a function which takes as input a vector v and a set of (non-canonical) basis vectors b_i and returns the coefficients of v in that basis.

7.1 Representing vectors as functions

In what follows we will systematically use the representation of vectors as a linear combination of basis vectors. There is a temptation to model the corresponding collection of coefficients as a list, or a tuple, but a more general (and conceptually simpler) way is to view them as a *function* from a set of indices G:

Nicola.Botta@pik-potsdam.de

```
newtype Vector s g = V (g → s) deriving (Additive, AddGroup)
```

We define, right away, the notation $a\,!\,i$ for the coefficient of the canonical basis vector $e\ i$, as follows:

```
infix 9 !
(!) :: Vector s g → g → s
V f ! i = f i
```

We sometimes omit the constructor V and the indexing operator $(!)$, thereby treating vectors as functions without the **newtype**. (We use the exclamation mark as an infix operator here as is common in programming, even though it is often used as postfix notation for factorial in mathematics texts.)

→ book!

As discussed above, the S parameter in *Vector S* has to be a field ($\mathbb{R}$, or $\mathbb{C}$, or $\mathbb{Z}_p$[1], etc.) for values of type *Vector S G* to represent elements of a vector space.

The cardinality of G, which we sometimes denote *card G*, is the number of basis vectors, and thus the dimension of the vector space. Often G is finite, and in the examples so far we have used indices from $G = \{0, \ldots, n\}$. Thus the dimension of the space would be $n + 1$.

In Haskell finiteness of G can be captured by the conjunction of *Bounded* (there is a minimum and a maximum element in G) and *Enum*erable (there is a notion of enumeration from a given element of G) and *Eq*. Hence, the list of all elements of G can be extracted:

```
type Finite g = (Bounded g, Enum g, Eq g)
finiteDomain :: Finite a ⇒ [a]
finiteDomain = [minBound .. maxBound]
```

→ live!

For our running example $finiteDomain = [0, 1, 2, 3, 4, 5, 6]$.

→ remember:

VectorSpace V S =
Field S ∧ AddGroup V ∧
(◁) ∧ ...

Let us now define a *VectorSpace* instance for the *Vector* representation. This can only be done if s is a *Field*. Then, we must provide an associative and commutative addition operation. For *Vector*, it can is defined indexwise. Because indexwise addition is already our definition of addition for functions $(g \to s)$, from Section 3.5.3, we can simply reuse this definition. (Function addition demands that s is an instance of *AddGroup*, but it's fine since s is even a *Field*.) This is what the **deriving** clause amounts to in the definition of **newtype** *Vector*. The rest of the *AddGroup* structure, *zero* and *negate* is defined by the same means.

What about vector scaling, $(\triangleleft)$? Can we simply reuse the definition that we had for functions? No, because multiplication of vectors does not work pointwise. In fact, attempting to lift multiplication from the *Multiplicative* class

[1] The set $\mathbb{Z}_p$ is the set of integers modulo p. We let the reader lookup the appropriate notion of division for it.

would give a homogeneous multiplication operator $(*) :: v \to v \to v$, but such an operator is not part of the definition of vector spaces. Consequently, vector spaces are in general *not* rings.

Indeed, the scaling operator $(\triangleleft) :: s \to v \to v$, is inhomogeneous: the first argument is a scalar and the second one is a vector. For our representation it can be defined as follows:

```
instance Field s ⇒ VectorSpace (Vector s g) s where (◁) = scaleV
scaleV :: Multiplicative s ⇒ s → Vector s g → Vector s g
scaleV s (V a) = V (λi → s * a i)
```

Exercise 7.2. Show that *Vector s g* satisfies the laws of vector spaces.

The canonical basis for *Vector* are given by

```
e :: (Eq g, Ring s) ⇒ g → Vector s g
e i = V (λj → i `is` j)
```

In linear algebra textbooks, the function *is* is often referred to as the Kronecker-delta function and *is i j* is written $\delta_{i,j}$.

```
is :: (Eq g, Ring s) ⇒ g → g → s
is i j = if i == j then one else zero
```

It is 1 if its arguments are equal and 0 otherwise. Thus *e i* has zeros everywhere, except at position *i* where it has a one.

We can see that, as expected, every $v : g \to s$ is a linear combination of vectors *e i* where the coefficient of the canonical basis vector *e i* is the scalar *v i*:

$$V\ v == (v\ 0 \triangleleft e\ 0) + \ldots + (v\ n \triangleleft e\ n)$$

This property is called the *characterising equation* for vectors.

To be sure, every vector *v* is a linear combination of any collection of basis vectors. But when using *canonical* basis vectors, the coefficients come simply from applying *v* (seen as a function) to the possible indices. Because we will work with many such linear combinations we introduce a helper function *linComb* for the right-hand side of the characterising equation:

```
linComb :: (Finite g, VectorSpace v s) ⇒ (g → s) → (g → v) → v
linComb v e = sum (map (λj → v j ◁ e j) finiteDomain)
```

where you can think of *finiteDomain* as enumerating the indices $[0\,..\,n]$.

Using *linComb* the characterising equation for vectors reads:

$$V\ v == linComb\ v\ e \tag{7.1}$$

Nicola.Botta@pik-potsdam.de

what guarantees that AddGroup (Vector s g)?

Live!

Nicola.Botta@pik-potsdam.de

Exercise 7.2: show that VectorSG satisfies the laws

We have to show that VectorSG as instance of AddGroup and with

$$s \triangleleft (V\ v) = V\ (\lambda g \to s * v\,g)$$

fulfills the vector space laws 1., 2. and 3.

a) remember (go back) to the VS laws

b) prove the claim

Exercise 7.2: show that VectorSG satisfies the laws

1) $\delta\Delta(Va + Vb) = \delta\Delta(Va) + \delta\Delta(Vb)$

$\delta\Delta(V(\ ?\ )) = V\ ?$

$\delta\Delta(neg(Va)) = neg(\delta\Delta(Va))$

2) ....

3) ....

Nicola.Botta@pik-potsdam.de

Exercise 7.2: show that Vector SG satisfies the laws

$$\partial\,\Delta\,(Va + Vb) = \text{def. of + for Vector SG}$$

$$\partial\,\Delta\,(V\,(\lambda g \to a\,g + b\,g)) = \cdots$$

$$\vdots$$

$$\partial\,\Delta\,(Va) + \partial\,\Delta\,(Vb) \qquad \square$$

would give a homogeneous multiplication operator $(*) :: v \to v \to v$, but such an operator is not part of the definition of vector spaces. Consequently, vector spaces are in general *not* rings.

Indeed, the scaling operator $(\triangleleft) :: s \to v \to v$, is inhomogeneous: the first argument is a scalar and the second one is a vector. For our representation it can be defined as follows:

```
instance Field s ⇒ VectorSpace (Vector s g) s where (◁) = scaleV
scaleV :: Multiplicative s ⇒ s → Vector s g → Vector s g
scaleV s (V a) = V (λi → s * a i)
```

Exercise 7.2. Show that *Vector s g* satisfies the laws of vector spaces.

The canonical basis for *Vector* are given by

```
e :: (Eq g, Ring s) ⇒ g → Vector s g
e i = V (λj → i `is` j)
```

In linear algebra textbooks, the function *is* is often referred to as the Kronecker-delta function and *is i j* is written $\delta_{i,j}$.

```
is :: (Eq g, Ring s) ⇒ g → g → s
is i j = if i == j then one else zero
```

It is 1 if its arguments are equal and 0 otherwise. Thus *e i* has zeros everywhere, except at position *i* where it has a one.

We can see that, as expected, every $v : g \to s$ is a linear combination of vectors *e i* where the coefficient of the canonical basis vector *e i* is the scalar *v i*:

$$V\ v == (v\ 0 \triangleleft e\ 0) + \ldots + (v\ n \triangleleft e\ n)$$

This property is called the *characterising equation* for vectors.

To be sure, every vector *v* is a linear combination of any collection of basis vectors. But when using *canonical* basis vectors, the coefficients come simply from applying *v* (seen as a function) to the possible indices. Because we will work with many such linear combinations we introduce a helper function *linComb* for the right-hand side of the characterising equation:

```
linComb :: (Finite g, VectorSpace v s) ⇒ (g → s) → (g → v) → v
linComb v e = sum (map (λj → v j ◁ e j) finiteDomain)
```

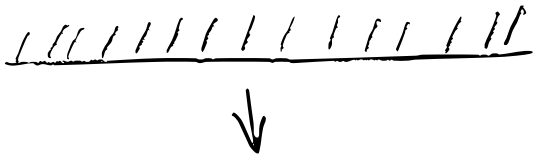
where you can think of *finiteDomain* as enumerating the indices $[0\,..\,n]$.

Using *linComb* the characterising equation for vectors reads:

```
V v == linComb v e
```

(7.1)

Nicola.Botta@pik-potsdam.de



what kind of equality? How can we see this?

V v = linComb v e

Nicola.Botta@pik-potsdam.de

Exercise 7.3. Using the elements defined above, sketch the isomorphism between an abstract vector space and its representation. Recall the definition of isomorphism in Section 4.2.3.

7.2 Linear transformations

As we have seen in earlier chapters, morphisms between structures are often important. Vector spaces are no different: if we have two vector spaces *Vector S G* and *Vector S G′* for the same set of scalars *S*, we can study functions $f : \mathit{Vector}\ S\ G \to \mathit{Vector}\ S\ G'$:

$$f\ (V\ v) = f\ (v\ 0 \triangleleft e\ 0 + \ldots + v\ n \triangleleft e\ n)$$

It is particularly interesting to study functions which preserve the vector space structure: vector-space homomorphisms. Such functions are more commonly called "linear maps", but to avoid unnecessary confusion with the Haskell *map* function we will refer to vector-space homomorphisms by the slightly less common name "linear transformation". Spelling out the homomorphism, the function f is a linear transformation if it maps the operations in *Vector S G* into operations in *Vector S G′* as follows:

$$\begin{aligned} f\ (u + v) &= f\ u + f\ v \\ f\ (s \triangleleft u) &= s \triangleleft f\ u \end{aligned}$$

Because $V\ v = \mathit{linComb}\ v\ e = (v\ 0 \triangleleft e\ 0 + \ldots + v\ n \triangleleft e\ n)$, we also have:

$$\begin{aligned} f\ (V\ v) &= f\ (v\ 0 \triangleleft e\ 0 \quad + \ldots + v\ n \triangleleft e\ n) && \text{\{- because } f \text{ is linear -\}} \\ &= \quad v\ 0 \triangleleft f\ (e\ 0) + \ldots + v\ n \triangleleft f\ (e\ n) && \text{\{- by def. of } \mathit{linComb} \text{ -\}} \\ &= \mathit{linComb}\ v\ (f \circ e) \end{aligned}$$

But this means that we can determine the whole function $f : \mathit{Vector}\ S\ G \to \mathit{Vector}\ S\ G'$ on all vectors from just f on the base vectors: $f \circ e : G \to \mathit{Vector}\ S\ G'$, which has a much smaller domain. Let $m = f \circ e$. Then, for each i, the vector $m\ i$ is the image of the canonical basis vector $e\ i$ through f. Then

$$f\ (V\ v) = \mathit{linComb}\ v\ m = v\ 0 \triangleleft m\ 0 + \ldots + v\ n \triangleleft m\ n$$

Each of the $m\ k$ is a *Vector S G′*, as is the resulting $f\ (V\ v)$. If we look at the component g' of $f\ (V\ v)$ we have

$$\begin{aligned} & f\ (V\ v)\ !\ g' && = \text{\{- as above -\}} \\ & (\mathit{linComb}\ v\ m)\ !\ g' && = \text{\{- } \mathit{linComb}, (\triangleleft), (+) \text{ are all linear -\}} \\ & \mathit{linComb}\ v\ (\lambda g \to m\ g\ !\ g') \end{aligned}$$

Nicola.Botta@pik-potsdam.de

Exercise 7.3: using the elements defined above, sketch the isomorphism between an abstract vector space and its representation. Recall the definition of isomorphism in Section 4.2.3

elements defined above:

- Vector S G ≅ G → S
- We already know a lot about G → S
- def. of ≅

definition of isomorphism from 4.2.3

Exercise 4.6. Show that *apply c* is an *Additive* homomorphism for all c, where $apply\ x\ f = f\ x$.

Solution: Indeed, writing $h = apply\ c$ for some fixed c, we have

$$
\begin{aligned}
&h\ (f+g) = \{\text{- def. } apply \text{ -}\}\\
&(f+g)\ c = \{\text{- def. } (+) \text{ for functions -}\}\\
&f\ c + g\ c = \{\text{- def. } apply \text{ -}\}\\
&h\ f + h\ g
\end{aligned}
$$

and

$$
\begin{aligned}
&h\ zero = \{\text{- def. } apply \text{ -}\}\\
&zero\ c = \{\text{- def. } zero \text{ for functions -}\}\\
&zero
\end{aligned}
$$

4.2.3 *Isomorphisms

Two homomorphisms which are inverse of each other define an *isomorphism*. If an isomorphism exists between two sets, we say that they are isomorphic. For example, the exponential and the logarithm witness an isomorphism between $\mathbb{R}_{>0}$ and $\mathbb{R}$.

Exercise 4.7. Show that exponential and logarithm are inverse of each other.

Exercise 4.8. Extend the *exp-log*-isomorphism to relate the methods of the *AddGroup* and *MulGroup* classes.

Exercise 4.9 (Hard.). Sketch the isomorphism between IPC and STLC seen in Section 2.1.6. What are the structures? What are mappings (functions)?

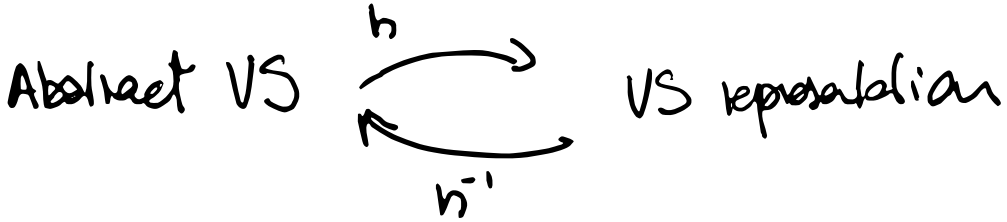
Exercise 4.10. Sketch an isomorphism between pairs of numbers and complex numbers, as suggested in Section 1.4.

4.3 Compositional semantics

The core of compositional semantics is that the meaning (semantics) of an expression is determined by combining the meanings of its subexpressions. Earlier, we have presented several DSLs as a syntax datatype *Syn* connected to a type *Sem* for the semantic values by a compositional semantic function $eval : Syn \to Sem$. In this section we show that compositional functions are homomorphisms and provide some examples with proofs of compositionality (or the opposite).

Nicola.Botta@pik-potsdam.de

Thus, we are asked to sketch 2 homomorph. which are "the inverse of each other" between:

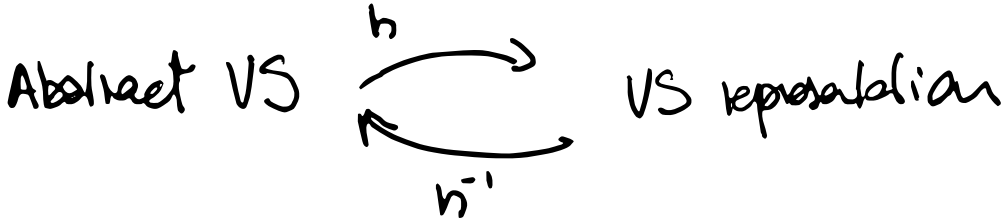


and preserve the vector space structure

- How do we proceed?

Nicola.Botta@pik-potsdam.de

Thus, we are asked to sketch 2 homomorph. which are "the inverse of each other" between:

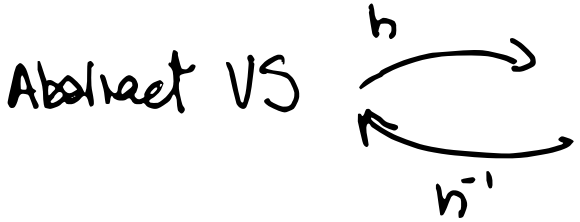


and preserve the vector space structure

- How do we proceed?
- What is this structure?

Nicola.Botta@pik-potsdam.de

Thus, we are asked to sketch 2 homomorph. which are "the inverse of each other" between:



and preserve the vector space structure

- How do we proceed?
- What is this structure?
  - L AddGroup + VS laws!

Nicola.Botta@pik-potsdam.de

- Start by naming the elements of the structure on both sides:

$zero_A$	$: V_A$	$zero$	$: G \to S$
$(+_A)$	$: V_A \to V_A \to V_A$	$(+)$	$: \cdots$
neg_A	$: V_A \to V_A$	neg	$: \cdots$
$(\triangleleft_A)$	$: S_A \to V_A \to V_A$	$(\triangleleft)$	$: \cdots$

- Make yourself clear what are the types of h, h^{-1}:

$h : ?$ $\qquad$ $h^{-1} : ?$

Nicola.Botta@pik-potsdam.de

- Make yourself clear what it means for h, h^{-1} to preserve the AddGroup + VS laws s/v.

$$h\ zero_A = zero$$

$$h\ (neg_A\ v) = neg\ (h\ v)$$

$$h\ (v +_A w) = h\ v + h\ v$$

$$h\ (s \cdot_A v) = \ ?$$

$$h^{-1} \ldots .$$

?

- What could h, h^{-1} possibly be ?

Nicola.Botta@pik-potsdam.de

- Remember $h : V_A \to (G \to S)$
- We have seen that if we select a basis $b_0 \dots b_n$ in V_A, we get a <u>representation</u> ρ_v for any vector v in V_A.
- This means that we have a function

$$\rho : V_A \to (\{0 \dots n\} \to S_A)$$

- Take $h = \rho$, $G = \{0 \dots n\}$, $S = S_A$
- $(h\,v\,0) \cdot_A b_0 + \dots + (h\,v\,n) \cdot_A b_n = v$

Nicola.Botta@pik-potsdam.de

- Take $h^{-1} f = (f\,0) \triangleleft_A b_0 + \cdots + (f\,n) \triangleleft_A b_n$

We have:

$h^{-1}(h\,v)$

$= \{$ spec. of $h^{-1}\}$

$(h\,v\,0) \triangleleft_A b_0 + \cdots + (h\,v\,n) \triangleleft_A b_n$

$= \{$ spec. of $h\}$

v

$\Rightarrow \quad h^{-1} \circ h = id$!

Nicola.Botta@pik-potsdam.de

- What about $h \circ h^{-1}$?
- We will have to show that h is a homomorph.

.... exercise session ?

Nicola.Botta@pik-potsdam.de

Exercise 7.3. Using the elements defined above, sketch the isomorphism between an abstract vector space and its representation. Recall the definition of isomorphism in Section 4.2.3.

7.2 Linear transformations

As we have seen in earlier chapters, morphisms between structures are often important. Vector spaces are no different: if we have two vector spaces *Vector S G* and *Vector S G′* for the same set of scalars *S*, we can study functions $f : \mathit{Vector}\ S\ G \to \mathit{Vector}\ S\ G'$:

$$f\ (V\ v) = f\ (v\ 0 \triangleleft e\ 0 + \ldots + v\ n \triangleleft e\ n)$$

It is particularly interesting to study functions which preserve the vector space structure: vector-space homomorphisms. Such functions are more commonly called "linear maps", but to avoid unnecessary confusion with the Haskell *map* function we will refer to vector-space homomorphisms by the slightly less common name "linear transformation". Spelling out the homomorphism, the function f is a linear transformation if it maps the operations in *Vector S G* into operations in *Vector S G′* as follows:

$$\begin{aligned} f\ (u + v) &= f\ u + f\ v \\ f\ (s \triangleleft u) &= s \triangleleft f\ u \end{aligned}$$

Because $V\ v = \mathit{linComb}\ v\ e = (v\ 0 \triangleleft e\ 0 + \ldots + v\ n \triangleleft e\ n)$, we also have:

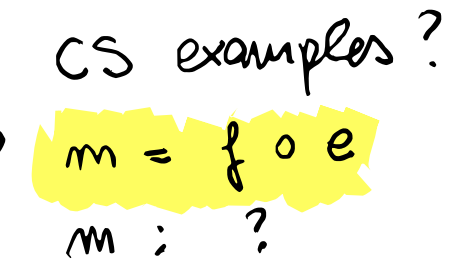
$$\begin{aligned} f\ (V\ v) &= f\ (v\ 0 \triangleleft e\ 0 \quad + \ldots + v\ n \triangleleft e\ n) && \text{\{- because } f \text{ is linear -\}} \\ &= \quad v\ 0 \triangleleft f\ (e\ 0) + \ldots + v\ n \triangleleft f\ (e\ n) && \text{\{- by def. of } \mathit{linComb} \text{ -\}} \\ &= \mathit{linComb}\ v\ (f \circ e) \end{aligned}$$

But this means that we can determine the whole function $f : \mathit{Vector}\ S\ G \to \mathit{Vector}\ S\ G'$ on all vectors from just f on the base vectors: $f \circ e : G \to \mathit{Vector}\ S\ G'$, which has a much smaller domain. Let $m = f \circ e$. Then, for each i, the vector $m\ i$ is the image of the canonical basis vector $e\ i$ through f. Then

$$f\ (V\ v) = \mathit{linComb}\ v\ m = v\ 0 \triangleleft m\ 0 + \ldots + v\ n \triangleleft m\ n$$

Each of the $m\ k$ is a *Vector S G′*, as is the resulting $f\ (V\ v)$. If we look at the component g' of $f\ (V\ v)$ we have

$$\begin{aligned} & f\ (V\ v)\ !\ g' && = \text{\{- as above -\}} \\ & (\mathit{linComb}\ v\ m)\ !\ g' && = \text{\{- } \mathit{linComb}, (\triangleleft), (+) \text{ are all linear -\}} \\ & \mathit{linComb}\ v\ (\lambda g \to m\ g\ !\ g') \end{aligned}$$



Nicola.Botta@pik-potsdam.de

C

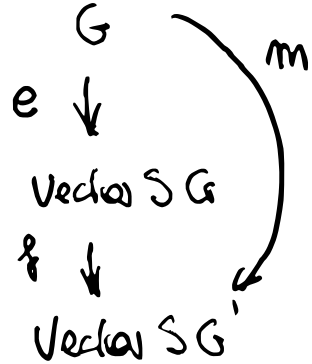
What does the (suspicious) equality

$$f\ (V\ v)\ !\ g' = \text{linComb}\ v\ (\lambda g \to m\ g\ !\ g')$$

mean? First:

$$f : \text{Vector}\ S\ G \to \text{Vector}\ S\ G'$$

$$m : G \to \text{Vector}\ S\ G'$$



$$\Rightarrow$$

$$\lambda g \to m\ g\ !\ g' : G \to S$$

This works because scalars are vectors!

Nicola.Botta@pik-potsdam.de

Second: if we spell out the composition $(m = f \circ e)$

$$f\,(V\,v) = v_0 \triangleleft m_0 + \cdots + v_n \triangleleft m_n =$$

linComb v m

we see that we are actually multiplying the matrix whose columns are $m_0 \cdots m_n$ with the vector $V\,v$. Define:

mulMV (trans m) (V v) = linComb v m

Nicola.Botta@pik-potsdam.de

```
trans : (G -> Vector S G') -> (G' -> Vector S G)
      : Matrix S G' G      -> Matrix S G G'
```

```
trans m = λg' -> V (λg -> m g ! g')
```

Example ($G = \{0,1\}$, $G' = \{0,1,2\}$):

$$m\,0 = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad m\,1 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad m = \begin{pmatrix} 1 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

M = transpose m

$$M\,0 = V(\lambda g \to m\,g\,!\,0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$
$$M\,1 = V(\lambda g \to m\,g\,!\,1) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$
$$M\,2 = V(\lambda g \to m\,g\,!\,2) = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

$$\text{transpose} \begin{pmatrix} 1 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix}$$

Nicola.Botta@pik-potsdam.de

```
trans m = λq' -> V (λq -> mq ! q')
```

→ ①

- trans ∘ trans = id

```
trans (trans m) k ! e = {??}
V (λq -> (trans m) q ! k) ! e = {??}
(trans m) e ! k = {??}
V (λq -> m q ! e) ! k = {??}
m k ! e
```

- `mulMV m (V v) = linComb v (trans m)` → ②

Nicola.Botta@pik-potsdam.de

That is, it suffices to know the behaviour of f on the basis vectors to know its behaviour on the whole vector space.

It is enlightening to compare the above sum with the standard vector-matrix multiplication. Let us define M as follows:

$$M = \begin{bmatrix} m\ 0 & \cdots & m\ n \end{bmatrix} \qquad \text{where } m : G \to \mathit{Vector}\ S\ G'$$

That is, the columns of M are $m\ 0$ to $m\ n$, or, in other words, the columns of M are $f\ (e\ i)$. Every $m\ k$ has $\mathit{card}\ G'$ elements, and it has become standard to write $M\ i\,!\,j$ to mean the ith element of the jth column, i.e., $M\ i\,!\,j = m\ j\,!\,i$, so that, if we denote the usual matrix-vector multiplication by mulMV:

$$(\mathit{mulMV}\ M\ (V\ v))\ i = \mathit{linComb}\ v\ (M\ i)$$

therefore, one has

$$\begin{aligned}
&(\mathit{mulMV}\ M\ (V\ v))\ i = && \text{-- by def. of } \mathit{mulMV}\\
&\mathit{linComb}\ v\ (M\ i) \quad = && \text{-- by def. of } (M\ i)\,!\,j\\
&\mathit{linComb}\ v\ (\lambda j \to m\ j\,!\,i) = && \text{-- earlier computation (linearity)}\\
&f\ (V\ v)\ i
\end{aligned}$$

If we take Matrix to be just a synonym for functions of type $G \to \mathit{Vector}\ S\ G'$:

$$\textbf{type}\ \mathit{Matrix}\ s\ g\ g' = g' \to \mathit{Vector}\ s\ g$$

then we can implement matrix-vector multiplication as follows:

$$\begin{aligned}
&\mathit{mulMV} :: (\mathit{Finite}\ g, \mathit{Field}\ s) \Rightarrow \mathit{Matrix}\ s\ g\ g' \to \mathit{Vector}\ s\ g \to \mathit{Vector}\ s\ g'\\
&\mathit{mulMV}\ m\ (V\ v) = \mathit{linComb}\ v\ (\mathit{transpose}\ m)\\
&\mathit{transpose} :: \mathit{Matrix}\ s\ i\ j \to \mathit{Matrix}\ s\ j\ i\\
&\mathit{transpose}\ m\ i = V\ (\lambda j \to m\ j\,!\,i)
\end{aligned}$$

In the terminology of the earlier chapters, we can see $\mathit{Matrix}\ s\ g\ g'$ as a type of syntax and the linear transformation (of type $\mathit{Vector}\ S\ G \to \mathit{Vector}\ S\ G'$) as semantics. With this view, mulMV is just another evaluation function from syntax to semantics. However, again, given a fixed basis we have an isomorphism rather than a mere homomorphism: for a given linear transformation, the matrix representation is unique. Below we often write just an infix $(*)$ for mulMV.

Example Consider the multiplication of a matrix with a basis vector:

$$(M * e\ k)\,!\,i = (\mathit{linComb}\ (\mathit{is}\ k)\ (\mathit{transpose}\ M))\,!\,i = M\ i\,!\,k$$

Nicola.Botta@pik-potsdam.de

? multMV M (V v) ! i = linComb v (M i) = f (V v) ! i

multMV M (V v) ! i =

linComb v (rows M) ! i =

(v 0 * (rows M) 0 + ... + v n * (rows M) n) ! i =

(v 0 * (rows M) 0) ! i + ... =

(v 0 * V (λ g -> M g ! 0)) ! i + ... =

....

but ... we do not need this because ....

Nicola.Botta@pik-potsdam.de

? mulHV M (V v) = {?}

linCom v (vars M) = {def. M}

linCom v (vars (vars m)) = {?}

linComb v m = {?}

f (V v)

Nicola.Botta@pik-potsdam.de

?? Example: mulMV m (ek) = trans m k

mulMV m (ek) = {??}

enComb (i,k) (trans m) = {??}

trans m k

Nicola.Botta@pik-potsdam.de

?? Example: `mulMV m (ek) = trans m k`

`mulMV m (ek)` = {?}

`linComb (i)k) (trans m)` = {?}

`trans m k`

Non

`mulMV m (ek) ! i` = {?}

`trans m k ! i` = {?}

`V (λg -> m g ! k) ! i` = {?}

`m i ! k`

Nicola.Botta@pik-potsdam.de

Recap L7.1 :

Nicola.Botta@pik-potsdam.de

Recap L7.1 :

introduced Vector S G ≅ G → S

V v = linComb v e : Vector S G

↳ : G → S ↳ can. basis : G → Vector S G

Nicola.Botta@pik-potsdam.de

Recap L7.1:

\# introduced Vector S G ≅ G → S

$v = \text{linComb}\ v\ e$: Vector S G

↳ : G → S ↳ can. basis : G → Vector S G

\# lin. map f : f : Vector S G → Vector S G'

$f(v) = \text{linComb}\ v\ (f \circ e)$

Nicola.Botta@pik-potsdam.de

Recap L7.1:

introduced Vector S G ≅ G → S

```
Vv = linComb v e : Vector S G
```

↳ : G → S ↳ can. basis : G → Vector S G

lin. transf : f : Vector S G → Vector S G'

```
f (Vv) = linComb v f∘e
```

trans : Matrix S G G' → Matrix S G' G

mulMV : Matrix S G G' → Vector S G → Vector S G'

```
f x = mulMV (trans (f∘e)) x
```

Nicola.Botta@pik-potsdam.de

Questions:

Can we express the result of applying a linear transf. f to a vector x as a linComb if the basis is non canonical?

Nicola.Botta@pik-potsdam.de

Questions:

Can we express the result of applying a linear transf. f to a vector x as a linComb if the basis is non canonical?

$$x = \text{linComb}\ e_x\ b\ ,\ f\ \text{lin.} \Rightarrow$$

$$fx = \text{linComb}\ e_x\ f \circ b$$

Nicola.Botta@pik-potsdam.de

Questions:

Why do we represent vectors as functions? Why not using lists?

Nicola.Botta@pik-potsdam.de

Questions:

Why do we represent vectors as functions? Why not using lists?

```
Vector S = [S]

instance Field S => VS [S] S where

  s .* v =
```

Nicola.Botta@pik-potsdam.de

Questions:

Why do we represent vectors as functions? Why not using lists?

```
Vector S = [S]

instance Field S => VS [S] S where
  s .* v = ...
```

but:

```
Matrix S = [[S]]

mulMV = [[S]] -> [S] -> [S]

mulMV : Matrix S G G' -> Vector S G -> Vector S G'
      ( = G' -> Vector S G
```

i.e., $e\ k$ extracts the kth column from M (hence the notation "e" for "extract").

We have seen how a linear transformation f can be fully described by a matrix of scalars, M. Similarly, in the opposite direction, given an arbitrary matrix M, we can define

$$f\ v = M * v$$

and obtain a linear transformation $f = (M*)$. Moreover $((M*) \circ e)\ g\ !\ g' = M\ g'\ !\ g$, i.e., the matrix constructed as above for f is precisely M.

In Exercise 7.9 you verify this by computing $((M*) \circ e)\ g\ !\ g'$.

Therefore, every linear transformation is of the form $(M*)$ and every $(M*)$ is a linear transformation. There is a bijection between these two sets. Matrix-matrix multiplication is defined in order to ensure associativity (note here the overloading of the operator $*$):

$$(M' * M) * v = M' * (M * v)$$

that is, if we abstract over the vector v on both sides:

$$((M' * M)*) = (M'*) \circ (M*)$$

You may want to refer to Exercise 7.10, which asks you to work this out in detail, and Exercise 7.11 is about associativity of matrix-matrix multiplication.

A simple vector space is obtained for $G = ()$, the singleton index set. In this case, the vectors $s : () \to S$ are functions that can take exactly one value as argument, therefore they have exactly one value: $s\ ()$, so they are isomorphic with S. But, for any $v : G \to S$, we have a function $fv : G \to (() \to S)$, namely

$$fv\ g\ () = v\ g$$

fv is similar to our m function above. The associated matrix is

$$M = [m\ 0 \quad \cdots \quad m\ n] = [fv\ 0 \quad \cdots \quad fv\ n]$$

having $n + 1$ columns (the dimension of *Vector* G) and one row (dimension of *Vector* $()$). Let $w :: G \to S$:

$$M * (V\ w) = w\ 0 \triangleleft fv\ 0 + \ldots + w\ n \triangleleft fv\ n$$

$M * (V\ w)$ and each of the $fv\ k$ are "almost scalars": functions of type $() \to S$, thus, the only component of $M * (V\ w)$ is

$$\begin{aligned}(M * (V\ w))\ () &= w\ 0 * fv\ 0\ () + \ldots + w\ n * fv\ n\ () \\ &= w\ 0 * v\ 0 + \ldots + w\ n * v\ n\end{aligned}$$

i.e., the scalar product of the vectors v and w.

Remark: We have not yet discussed the geometrical point of view.

Nicola.Botta@pik-potsdam.de

End L1

→ Exercise 7.9

→ Extra !

(*): Matrix S G G' →
Vector S G →
Vector S G'

Nicola.Botta@pik-potsdam.de

Exercise 7.9 : $((M*) \circ e)\, g\, !\, g' = M\, g'\, !\, g$

M : Matrix S G G' = G' → Vector S G

e : ?

$(M*) = \text{mulMV}\ M$: ?

$(M*) \circ e$: ?

$((M*) \circ e)\, g\, !\, g' =$ {?}

$((M*)\,(e\, g))\, !\, g' =$

.....

L 7.1.4 !

mulMV m (ek) ! i = {?}
Haus m k ! i = {?}
V (λg → m g ! k) ! i = {?}
m i ! k

Nicola.Botta@pik-potsdam.de

Exercise 7.9 : $((M*) \circ e)\, q\, !\, q' = M\, q'\, !\, q$

is meant to demonstrate that "the matrix associated to $(M*)$" is in fact M ! How we achieved this ?

trans $((M*) \circ e)$ = { trans m = λq' -> V (λq -> mq ! q') }

$\lambda q' \to V(\lambda q \to ((M*) \circ e)\, q\, !\, q') =$

$\lambda q' \to V(\lambda q \to M\, q'\, !\, q) =$

$\lambda q' \to V(M\, q'\, !) =$ {?}

$\lambda q' \to M\, q' =$ M

Nicola.Botta@pik-potsdam.de

$\lambda g' \to V(Mg'!) = \{?\}$

$\lambda g' \to Mg' = M$

Prove: $\forall\, v : \text{Vector SG}.\quad V(v!) = v$

Nicola.Botta@pik-potsdam.de

$\lambda g' \to V(Mg'!) = \{?\}$

$\lambda g' \to Mg' = M$

Prove: $\forall\, v : \text{Vector } SG.\ V(v!) = v$

This is trivial:

$V(v!)!k = \{\ j\ \}$

$v!k$

but I want the right justification j !

Nicola.Botta@pik-potsdam.de

Extra: (Lin. transf. and Lin. sys. of eqs.):

Let $\rho_v : G \to S$ be the unique repr. of $v : \text{Vector } S\ G$ in $b : G \to \text{Vector } S\ G$:

$$v = \text{linComb}\ \rho_v\ b$$

Show that $V\rho_v$ is the (unique) solution of

$$(\text{trans}\ b) * (V\rho_v) = v$$

How does this relate to Exercise 7.1?

Nicola.Botta@pik-potsdam.de

i.e., $e\ k$ extracts the kth column from M (hence the notation "e" for "extract").

We have seen how a linear transformation f can be fully described by a matrix of scalars, M. Similarly, in the opposite direction, given an arbitrary matrix M, we can define

$$f\ v = M * v$$

and obtain a linear transformation $f = (M*)$. Moreover $((M*) \circ e)\ g\ !\ g' = M\ g'\ !\ g$, i.e., the matrix constructed as above for f is precisely M.

In Exercise 7.9 you verify this by computing $((M*) \circ e)\ g\ !\ g'$.

Therefore, every linear transformation is of the form $(M*)$ and every $(M*)$ is a linear transformation. There is a bijection between these two sets. Matrix-matrix multiplication is defined in order to ensure associativity (note here the overloading of the operator $*$):

$$(M' * M) * v = M' * (M * v)$$

that is, if we abstract over the vector v on both sides:

$$((M' * M)*) = (M'*) \circ (M*)$$

You may want to refer to Exercise 7.10, which asks you to work this out in detail, and Exercise 7.11 is about associativity of matrix-matrix multiplication.

→ Exercise 7.10
→ Exercise 7.11

A simple vector space is obtained for $G = ()$, the singleton index set. In this case, the vectors $s : () \to S$ are functions that can take exactly one value as argument, therefore they have exactly one value: $s\ ()$, so they are isomorphic with S. But, for any $v : G \to S$, we have a function $fv : G \to (() \to S)$, namely

$$fv\ g\ () = v\ g$$

fv is similar to our m function above. The associated matrix is

$$M = \begin{bmatrix} m\ 0 & \cdots & m\ n \end{bmatrix} = \begin{bmatrix} fv\ 0 & \cdots & fv\ n \end{bmatrix}$$

having $n + 1$ columns (the dimension of *Vector* G) and one row (dimension of *Vector* $()$). Let $w :: G \to S$:

$$M * (V\ w) = w\ 0 \triangleleft fv\ 0 + \ldots + w\ n \triangleleft fv\ n$$

$M * (V\ w)$ and each of the $fv\ k$ are "almost scalars": functions of type $() \to S$, thus, the only component of $M * (V\ w)$ is

$$\begin{aligned} (M * (V\ w))\ () &= w\ 0 * fv\ 0\ () + \ldots + w\ n * fv\ n\ () \\ &= w\ 0 * v\ 0 + \ldots + w\ n * v\ n \end{aligned}$$

i.e., the scalar product of the vectors v and w.

Remark: We have not yet discussed the geometrical point of view.

Nicola.Botta@pik-potsdam.de

Exercise 7.10: mulMM is defined in order to ensure a homomorph. from (**) to (∘):

$$\forall M.\ \forall M'.\quad (M ** M')* = (M*) \circ (M'*)$$

That is $H_2(?, ?, (\circ))$.

1) Work out the types ...

2) Expand the definitions ...

Notice that we haven't already defined (**)!

Nicola.Botta@pik-potsdam.de

Remember $M * v = mulMV\ M\ v$. Thus

(*) : ?

(**) : ? $\rightarrow (G' \rightarrow Vec\ S\ G) \rightarrow$?

(o) : ?

What is here the homomorphism?

Between which sets?

Nicola.Botta@pik-potsdam.de

Exercise 7.11: Show that mulMM is assoc.

$$(A ** B) ** C = A ** (B ** C)$$

$$((A ** B) ** C) * v =$$

. . . . . .

$$(A ** (B ** C)) * v$$

Nicola.Botta@pik-potsdam.de

i.e., $e\ k$ extracts the kth column from M (hence the notation "e" for "extract").

We have seen how a linear transformation f can be fully described by a matrix of scalars, M. Similarly, in the opposite direction, given an arbitrary matrix M, we can define

$$f\ v = M * v$$

and obtain a linear transformation $f = (M*)$. Moreover $((M*) \circ e)\ g\ !\ g' = M\ g'\ !\ g$, i.e., the matrix constructed as above for f is precisely M.

In Exercise 7.9 you verify this by computing $((M*) \circ e)\ g\ !\ g'$.

Therefore, every linear transformation is of the form $(M*)$ and every $(M*)$ is a linear transformation. There is a bijection between these two sets. Matrix-matrix multiplication is defined in order to ensure associativity (note here the overloading of the operator $*$):

$$(M' * M) * v = M' * (M * v)$$

that is, if we abstract over the vector v on both sides:

$$((M' * M)*) = (M'*) \circ (M*)$$

You may want to refer to Exercise 7.10, which asks you to work this out in detail, and Exercise 7.11 is about associativity of matrix-matrix multiplication.

A simple vector space is obtained for $G = ()$, the singleton index set. In this case, the vectors $s : () \to S$ are functions that can take exactly one value as argument, therefore they have exactly one value: $s\ ()$, so they are isomorphic with S. But, for any $v : G \to S$, we have a function $fv : G \to (() \to S)$, namely

$$fv\ g\ () = v\ g$$

fv is similar to our m function above. The associated matrix is

$$M = [m\ 0 \quad \cdots \quad m\ n] = [fv\ 0 \quad \cdots \quad fv\ n]$$

having $n + 1$ columns (the dimension of *Vector* G) and one row (dimension of *Vector* $()$). Let $w :: G \to S$:

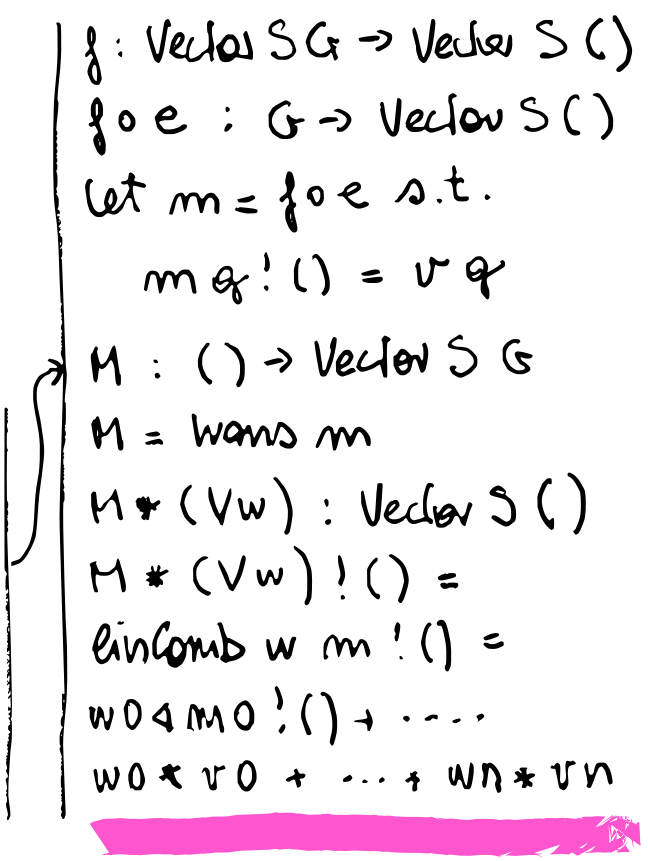
$$M * (V\ w) = w\ 0 \triangleleft fv\ 0 + \ldots + w\ n \triangleleft fv\ n$$

$M * (V\ w)$ and each of the $fv\ k$ are "almost scalars": functions of type $() \to S$, thus, the only component of $M * (V\ w)$ is

$$\begin{aligned}(M * (V\ w))\ () &= w\ 0 * fv\ 0\ () + \ldots + w\ n * fv\ n\ () \\ &= w\ 0 * v\ 0 + \ldots + w\ n * v\ n\end{aligned}$$

i.e., the scalar product of the vectors v and w.

Remark: We have not yet discussed the geometrical point of view.



Nicola.Botta@pik-potsdam.de

Extra on linear transform.

Certain linear transformations can also be interpreted as changes of basis (Ex. 7.1!)

Let

$$f : \text{Vector } S\ G \to \text{Vector } S\ G$$

such that

$$\forall u : G \to \text{Vector } S\ G.\ u \text{ basis} \Rightarrow f \circ u \text{ basis}$$

Let b be basis and ρ_v^b repr. of v in b:

Nicola.Botta@pik-potsdam.de

Extra on linear transform.

$$v = \text{lincomb}\ \rho_v^b\ b = (\text{trans}\ b) * (V\ \rho_v^b)$$

Because $b' = f \circ b$ is also a basis, we also have a unique repr. of v in b':

$$v = \text{lincomb}\ \rho_v^{b'}\ b' = (\text{trans}\ b') * (V\ \rho_v^{b'})$$

Thus

$$(\text{trans}\ b') * V(\rho_v^{b'}) = (\text{trans}\ b) * V(\rho_v^b)$$

Nicola.Botta@pik-potsdam.de

7.3 Inner products

An important concept is the inner product between vectors. We define inner product space as a vector space equipped with an inner product, as follows:

```
class VectorSpace v s ⇒ InnerSpace v s where
  inner :: v → v → s
```

Inner products have (at least) two aspects. First, they yield a notion of how "big" a vector is, the *norm*.

```
sqNorm :: InnerSpace v s ⇒ v → s
sqNorm v = inner v v
norm :: (InnerSpace v a, Algebraic a) ⇒ v → a
norm v = √(sqNorm v)
```

Additionally, the inner product often serves as a measure of how much vectors are similar to (or correlated with) each other.

For two non-zero vectors u and v, we can define:

```
similarity u v = inner u v / norm u / norm v
```

Dividing by the norms mean that $abs\ (similarity\ u\ v)$ is at most 1 — the similarity is always in the interval $[-1, 1]$.

For example, in Euclidean spaces, one defines the inner product to be the product of the cosine of the angle between the vectors and their norms. Consequently, *similarity* is the cosine of the angle between vectors.

For this reason, one says that two vectors are orthogonal when their inner product is 0 — even in non-Euclidean spaces.

Dot product An often used inner product is the dot product, defined as follows:[2]

```
dot :: (Field s, Finite g) ⇒ Vector s g → Vector s g → s
dot (V v) (V w) = linComb v w
```

We should note that the dot product acts on the representations (syntax). This means that it will *change* depending on the basis chosen to represent vectors. Thus, the dot product is a syntactic concept, and it should be clearly identified

[2] This code is using the one-dimensional vector space instance defined in Chapter 7.

why does this work?

Nicola.Botta@pik-potsdam.de

as such. This can be somewhat counterintuitive, because so far in this chapter it was fine to use representations (they were unique given the basis). To further confuse matters, in Euclidean spaces (which are often used as illustration) if the basis vectors are orthogonal, then the dot product coincides with the inner product. But, according to our methodology, one should start by defining a suitable inner product, and then check if the dot product is equivalent to it. See Section 7.4.3 for an example.

Orthogonal transformations An important subclass of the linear transformations are those which preserve the inner product.

$$inner\ (f\ u)\ (f\ v) = inner\ u\ v$$

In Euclidean spaces, such a transformation preserve angles. In the context of linear algebra they are either called orthogonal transformations (emphasising the preservation of angles) or unitary transformations (emphasising preservation of norms).[3]

Exercise 7.4. Can you express this condition as a homomorphism condition?

→ Exercise 7.4

Such transformations necessarily preserve the dimension of the space (otherwise at least one basis vector would be squished to nothing and inner products involving it become zero). The corresponding matrices are square.

Exercise 7.5. Prove that orthogonal operators form a monoid with multiplication as an operator.

→ Exercise 7.5

If angles are preserved what about distances? An isometry f is a distance-preserving transformation:

$$norm\ (f\ v) = norm\ v$$

We can prove that f is orthogonal iff. it is an isometry. The proof in the left-to-right direction is easy and left as an exercise. In the other direction one uses the equality:

→ Exha

$$4 * inner\ u\ v = sqNorm\ (u + v) - sqNorm\ (u - v)$$

In Euclidean spaces, this means that preserving angles and preserving distances go hand-in-hand.

[3] In today's mathematical vocabulary, the word "unitary" signals that a complex scalar field is used, whereas the word "orthogonal" signals that a real field is used, and that the space is Euclidean.

Nicola.Botta@pik-potsdam.de

Exercise 7.4: can you express

$$\text{inner}\ (f\,u)\ (f\,v) = \text{inner}\ u\ v \qquad (1)$$

as a homomorphism condition?

Remember 4.2.1: $h : A \to B$ homomorph. from $\oplus : A \to A \to A$ to $\otimes : B \to B \to B$ iff

$$\forall u.\ \forall v.\ h\,(u \oplus v) = (h\,u) \otimes (h\,v)$$

AKA $H_2(h, \oplus, \otimes)$. Can you express (1) as $H_2(?, ?, ?)$? Yes (how?) No (why?)

Nicola.Botta@pik-potsdam.de

Exercise 7.5: Prove that orth. operators form a monoid with multiplication as an operator

Let $OTSG$ be the orth. transf. on $VectorSG$:

$$OTSG = \{ f : ? \mid \forall u, v.\ in\,(fu)\,(fv) = in\,u\,v \}$$

Then $(OTSG, id, \circ)$ is a monoid:

- $\forall f, g : OTSG.\ f \circ g : OTSG$?
- $\forall f : OTSG.\ id \circ f = f \circ id = f$ ✓
- $\forall f, g, h : OTSG.\ f \circ (g \circ h) = (f \circ g) \circ h$ ✓

Nicola.Botta@pik-potsdam.de

- $\forall f, g : \mathrm{OTSG} \,.\, f \circ g : \mathrm{OTSG}$?

$\forall u, v. \quad \mathrm{in}\,((f \circ g)\,u)\,((f \circ g)\,v) = \mathrm{in}\,u\,v$

$\mathrm{in}\,((f \circ g)\,u)\,((f \circ g)\,v) = \{?\}$

.....

$\mathrm{in}\,u\,v$

Task: complete the proof & extend to OMSG !

Nicola.Botta@pik-potsdam.de

Extra: f orth. iff f isometry

- f orth. $\Rightarrow$ f isometry

$$\langle f u, f v \rangle = \langle u, v \rangle \quad \Rightarrow \quad \{?\}$$

$$\langle f u, f u \rangle = \langle u, u \rangle \quad = \quad \{?\}$$

$$\mathrm{sqNorm}(f u) = \mathrm{sqNorm}\, u \quad \Rightarrow \quad \{?\}$$

$$\sqrt{\mathrm{sqNorm}(f u)} = \sqrt{\mathrm{sqNorm}\, u} \quad = \quad \{?\}$$

$$\mathrm{norm}(f u) = \mathrm{norm}\, u$$

Nicola.Botta@pik-potsdam.de

Extra: f orth. **iff** f isometry

- f isometry $\Rightarrow$ f orthogonal

$$\left.\begin{array}{l} \text{norm}\,(f\,(u+v)) = \text{norm}\,(u+v) \;\wedge \\ \text{norm}\,(f\,(u-v)) = \text{norm}\,(u-v) \end{array}\right\} \Rightarrow$$

$$\left.\begin{array}{l} \text{sqNorm}\,(f\,(u+v)) = \text{sqNorm}\,(u+v) \;\wedge \\ \text{sqNorm}\,(f\,(u-v)) = \text{sqNorm}\,(u-v) \end{array}\right\} \Rightarrow$$

$$\text{sqNorm}\,(f\,(u+v)) - \text{sqNorm}\,(f\,(u-v))$$

$$=$$

$$\text{sqNorm}\,(u+v) - \text{sqNorm}\,(u-v) \qquad =$$

$$\ldots$$

$$\text{inner}\,(f\,u)\,(f\,v) = \text{inner}\,u\,v$$

?

Nicola.Botta@pik-potsdam.de

Orthogonal transformations enjoy many more useful properties — we have barely scratched the surface here. Among others, their rows (and columns) are orthogonal to each other. The are also invertible (and so they form a group), and the inverse is the given by (conjugate-) transpose of the matrix.

7.4 Examples of matrix algebra

7.4.1 Functions

A useful example of a vector space is the functions from $\mathbb{R}$ to $\mathbb{R}$. In terms of a *VectorSpace* instance we have:

```
instance VectorSpace (ℝ → ℝ) ℝ where
  s ◁ f = (s*) ∘ f
```

Here $s \triangleleft f$ scales the function f by s pointwise.

Exercise 7.6. Verify the *VectorSpace* laws for the above instance.

→ Exercise 7.6

D (s ◁ f) =

D ((s*) ∘ f) =

....

s ◁ Df ?

An example of a linear transformation is the derivative. Indeed, we have already seen that $D\ (f + g) = D\ f + D\ g$. The equation $D\ (s \triangleleft f) = s \triangleleft D\ f$ is verified by expanding the definitions. Together, this means that the laws of linear transformations are verified.

7.4.2 Polynomials and their derivatives

In Chapter 5, we have represented polynomials of degree $n + 1$ by the list of their coefficients. This is the same representation as the vectors represented by $n + 1$ coordinates which we referred to in the introduction to this chapter. Indeed, polynomials of degree n form a vector space, and we could interpret that as $\{0, \ldots, n\} \to \mathbb{R}$ (or, more generally, *Field* $a \Rightarrow \{0, \ldots, n\} \to a$). The operations, $(+)$ for vector addition and $(\triangleleft)$ for vector scaling, are defined in the same way as they are for functions.

To give an intuition for the vector space it is useful to consider the interpretation of the canonical basis vectors. Recall that they are:

$$e\ i : \{0, \ldots, n\} \to \mathbb{R}; e\ i\ j = i\ \text{`is'}\ j$$

but how do we interpret them as polynomial functions?

Nicola.Botta@pik-potsdam.de

Exercise 7.6 : verify the VS laws for $\mathbb{R} \to \mathbb{R}$

1. $s \triangleleft (f + g) = s \triangleleft f + s \triangleleft g$

   $s \triangleleft \text{zero} = \text{zero}$

   $s \triangleleft (\text{neg}\ f) = \text{neg}\ (s \triangleleft f)$

2. $(x + y) \triangleleft f = x \triangleleft f + y \triangleleft f$

   $\text{zero} \triangleleft f = \text{zero}$

   $(\text{neg}\ x) \triangleleft f = \text{neg}\ (x \triangleleft f)$

3. $(\triangleleft)\ \text{one} = \text{id}$

   $(\triangleleft)\ (x * y) = (\triangleleft)\ x \circ (\triangleleft)\ y$

Nicola.Botta@pik-potsdam.de

Exercise 7.6: verify the VS laws for $\mathbb{R} \to \mathbb{R}$

$$(\triangleleft)\,(x * y) = (\triangleleft)\,x \circ (\triangleleft)\,y$$

$$\begin{aligned}
&(\triangleleft)\,(x * y)\,f && = \{?\}\\
&(x * y) \triangleleft f && = \{?\}\\
&\lambda z \to (x * y) * (f\,z) && = \{?\}\\
&\lambda z \to x * (y * (f\,z)) && = \{?\}\\
&\lambda z \to x * ((y \triangleleft f)\,z) && = \{?\}\\
&x \triangleleft (y \triangleleft f) && = \{?\}\\
&(\triangleleft)\,x\,((\triangleleft)\,y\,f) && = \{?\}\\
&((\triangleleft)\,x \circ (\triangleleft)\,y)\,f
\end{aligned}$$

Nicola.Botta@pik-potsdam.de

Orthogonal transformations enjoy many more useful properties — we have barely scratched the surface here. Among others, their rows (and columns) are orthogonal to each other. The are also invertible (and so they form a group), and the inverse is the given by (conjugate-) transpose of the matrix.

7.4 Examples of matrix algebra

7.4.1 Functions

A useful example of a vector space is the functions from $\mathbb{R}$ to $\mathbb{R}$. In terms of a *VectorSpace* instance we have:

```
instance VectorSpace (ℝ → ℝ) ℝ where
  s ◁ f = (s*) ∘ f
```

Here $s \triangleleft f$ scales the function f by s pointwise.

Exercise 7.6. Verify the *VectorSpace* laws for the above instance.

An example of a linear transformation is the derivative. Indeed, we have already seen that $D\ (f + g) = D\ f + D\ g$. The equation $D\ (s \triangleleft f) = s \triangleleft D\ f$ is verified by expanding the definitions. Together, this means that the laws of linear transformations are verified.

7.4.2 Polynomials and their derivatives

In Chapter 5, we have represented polynomials of degree $n + 1$ by the list of their coefficients. This is the same representation as the vectors represented by $n + 1$ coordinates which we referred to in the introduction to this chapter. Indeed, polynomials of degree n form a vector space, and we could interpret that as $\{0, \ldots, n\} \to \mathbb{R}$ (or, more generally, *Field a* $\Rightarrow \{0, \ldots, n\} \to a$). The operations, $(+)$ for vector addition and $(\triangleleft)$ for vector scaling, are defined in the same way as they are for functions.

To give an intuition for the vector space it is useful to consider the interpretation of the canonical basis vectors. Recall that they are:

$$e\ i : \{0, \ldots, n\} \to \mathbb{R}; e\ i\ j = i \text{ `is'} j$$

but how do we interpret them as polynomial functions?

When we represented a polynomial by its list of coefficients in Chapter 5, we saw that the polynomial function $\lambda x \to x\hat{}3$ could be represented as $[0,0,0,1]$, where 1 is the coefficient of $x\hat{}3$.

This representation suggests to use as canonical basis vectors $e\ i$ the monomials $\lambda x \to x\hat{}i$. Representing the above list of coefficients as a vector is then a matter of converting lists to functions $\{0,\ldots,n\} \to \mathbb{R}$). This way, the vector $\lambda j \to$ **if** $j == 3$ **then** 1 **else** 0 is equal to $\lambda j \to 3$ '*is*' j or simply $e\ 3$. Any other polynomial function p equals the linear combination of monomials, and can therefore be represented as a linear combination of our basis vectors $e\ i$. For example, $p\ x = 2 + x\hat{}3$ is represented by $2 \lhd e\ 0 + e\ 3$.

The evaluator from the *Vector s g* representation to polynomial functions is as follows:

$$\begin{aligned} &\mathit{evalP} :: \mathit{Vector}\ \mathbb{R}\ \{0,\ldots,n\} \to (\mathbb{R} \to \mathbb{R}) \\ &\mathit{evalP}\ (V\ v)\ x = \mathit{sum}\ (\mathit{map}\ (\lambda i \to v\ i * x\hat{}i)\ [0\,..\,n]) \end{aligned}$$

Let us now turn to the representation of the derivative of polynomials. We have already seen in the previous section that the *derive* function is a linear transformation. We also know that it takes polynomials of degree $n+1$ to polynomials of degree n, and as such it is well defined as a linear transformation of polynomials too. Its representation can be obtained by applying the linear transformation to every basis vector:

$$M = [\mathit{derive}\ (e\ 0), \mathit{derive}\ (e\ 1), \ldots, \mathit{derive}\ (e\ n)]$$

where each $\mathit{derive}\ (e\ i)$ has length n. The vector $e\ (i+1)$ represents $\lambda x \to x\hat{}(i+1)$ and thus we want $\mathit{derive}\ (e\ (i+1))$ to represent the derivative of $\lambda x \to x\hat{}(i+1)$:

$$\begin{aligned} &\mathit{evalP}\ (\mathit{derive}\ (e\ (i+1))) = \{\text{- by spec. -}\} \\ &D\ (\mathit{evalP}\ (e\ (i+1))) \quad = \{\text{- by def. of } e, \mathit{evalP}\ \text{-}\} \\ &D\ (\lambda x \to x\hat{}(i+1)) \quad = \{\text{- derivative of a monomial -}\} \\ &\lambda x \to (i+1) * x\hat{}i \quad = \{\text{- by def. of } e, \mathit{evalP}, (\lhd)\ \text{-}\} \\ &\mathit{evalP}\ ((i+1) \lhd (e\ i)) \end{aligned}$$

Thus

$$\mathit{derive}\ (e\ (i+1)) = (i+1) \lhd (e\ i)$$

Also, the derivative of $\mathit{evalP}\ (e\ 0) = \lambda x \to 1$ is $\lambda x \to 0$ and thus $\mathit{derive}\ (e\ 0)$ is the zero vector:

$$\mathit{derive}\ (e\ 0) = 0$$

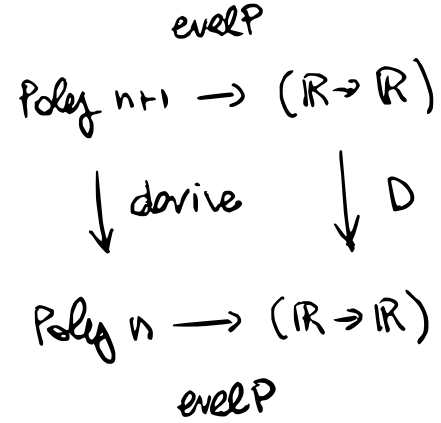
Nicola.Botta@pik-potsdam.de

e k repr. $\lambda x \to x^k$

How does the code look like?

→ Eine!

→ standard pattern!



Example: $n + 1 = 3$:

$$M = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Take the polynomial function $p\ x = 1 + 2 * x + 3 * x\hat{}2$ as a vector

$$v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

and we have

$$M * v = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 2 \end{bmatrix} \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \end{bmatrix}$$

representing the polynomial function $p'\ x = 2 + 6 * x$.

As an interesting follow-up, Exercise 7.12 asks you to write the (infinite-dimensional) matrix representing D for power series. Similarly, in Exercise 7.13 you compute the matrix In associated with integration of polynomials.

7.4.3 *Inner product for functions and Fourier series

We said before that the inner product yields a notion of norm and similarity. Can we use the dot product as inner product for power series (if the basis is $e\ i = x\hat{}i$)? We could, but then it would not be very useful. For example, it would not yield a useful notion of similarity between the represented function. To find a more useful inner product, we can return to the semantics of power series in terms of functions. But for now we consider them over the restricted domain $I = [-\pi, \pi]$.

Assume for a moment that we would define the inner product of functions u and v as follows:

$$innerF\ u\ v = \int_I (eval\ u\ x) * (eval\ v\ x)\ dx$$

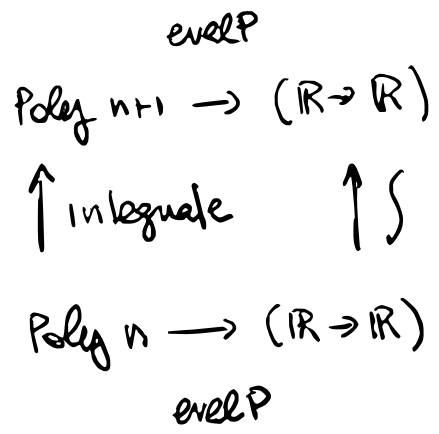
Then, the norm of a function would be a measure of how far it gets from zero, using a quadratic mean. Likewise, the corresponding similarity measure corresponds to how much the functions "agree" on the interval. That is, if the signs of $eval\ u$ and $eval\ v$ are the same on a sub-interval I then the integral is positive on I, and negative if they are different.

As we suspected, using $inner = innerF$, the straightforward representation of polynomials as list of coefficients is not an orthogonal basis. There is, for example, a positive correlation between the canonical vectors x and $x\hat{}3$.

Nicola.Botta@pik-potsdam.de

→ Exercise 7.12 Gie?

→ Exercise 7.13



If we were instead using a set of basis polynomials b_n which are orthogonal using the above definition of *inner*, then we could simply let $inner = dot$, and this would be a lot more efficient than to compute the integral by the following series of steps: 1) compute the product using *mulPoly*, 2) integrate using *integ*, 3) use *eval* on the end points of the domain.

Let us consider as a basis the functions $b_n\ x = sin\ (n * x)$, and prove that they are orthogonal.

We first use trigonometry to rewrite the product of basis vectors (call it $f_{ij}\ x$):

$$\begin{aligned} & f_{ij}\ x \\ &= 2 * (b_i\ x * b_j\ x) \\ &= 2 * sin\ (i * x) * sin\ (j * x) \\ &= cos\ ((i - j) * x) - cos\ ((i + j) * x) \end{aligned}$$

Assuming $i \neq j$, we can take the indefinite integral, and find (call it $F_{ij}\ x$):

$$F_{ij}\ x = sin\ ((i - j) * x)\ /\ (i - j) - sin\ ((i + j) * x)\ /\ (i + j) + K$$

Note that we only need the value of this function at the interval end-points: $-\pi$ and π. But $sin\ (k * \pi) = 0$ for any integer k, which means that $F_{ij}\ (-\pi) = F_{ij}\ \pi = K$. Taking the definite integral over the domain I yields:

$$2 * (inner\ b_i\ b_j) = F_{ij}\ \pi - F_{ij}\ (-\pi) = K - K = 0$$

and thus $inner\ b_i\ b_j = 0$

We can now compute *sqNorm* of b_i. Trigonometry says:

$$\begin{aligned} & 2 * (b_i\ x * b_i\ x) \\ &= 2 * sin\ (i * x) * sin\ (i * x) \\ &= cos\ (0 * x) - cos\ (2 * i * x) \\ &= 1 - cos\ (2 * i * x) \end{aligned}$$

When taking the integral on I, the cosine disappears using the same argument as before, and there remains: $2 * sqNorm\ b_i = 2\ \pi$. Thus to normalise the basis vectors we need to scale them by $1\ /\ \sqrt{\pi}$. In sum $b_i\ x = sin\ (i * x)\ /\ \sqrt{\pi}$ is an orthonormal basis:

$$b_i\ `innerF`\ b_j = is\ i\ j$$

As interesting as it is, this basis does not cover all functions over I. To start, $eval\ b_i\ 0 == 0$ for every i, and thus linear combinations can only ever be zero at the origin.

Nicola.Botta@pik-potsdam.de

But if we were to include $cos\ (n * x)\ /\ \sqrt{\pi}$ in the set of basis vectors, it would remain orthogonal, and the space would cover all periodic functions with period 2π. A representation of a function in this basis is called the Fourier series. Let us define a meaningful index (G) for the basis:

```
data Periodic where
  Sin :: ℕ>0 → Periodic
  Cos :: ℕ   → Periodic
  deriving Eq
```

For example, the function $f\ x = 3 * sin\ x + cos\ (2 * x) - 1$ is represented by the vector $v = 3 \triangleleft e\ (Sin\ 1) + e\ (Cos\ 2) - e\ (Cos\ 0)$ which can also be written:

```
v :: Vector ℝ Periodic
v = V vf
  where vf (Sin 1) = 3
        vf (Cos 2) = 1
        vf (Cos 0) = −1
        vf _       = 0
```

A useful property of an orthonormal basis is that its representation as coefficients can be obtained by taking the inner product with each basis vectors. Indeed, using Eq. (7.1) as a starting point, we can calculate:[4]

$$
\begin{aligned}
& v && == linComb\ v\ b \\
\Rightarrow\ & v && == sum\ [v_i \triangleleft b_i \mid i \leftarrow finiteDomain] \\
\Rightarrow\ & v\ `inner`\ b_j && == sum\ [v_i \triangleleft b_i \mid i \leftarrow finiteDomain]\ `inner`\ b_j \\
\Rightarrow\ & v\ `inner`\ b_j && == sum\ [v_i \triangleleft (b_i\ `inner`\ b_j) \mid i \leftarrow finiteDomain] \\
\Rightarrow\ & v\ `inner`\ b_j && == sum\ [v_i \triangleleft is\ i\ j \mid i \leftarrow finiteDomain] \\
\Rightarrow\ & v\ `inner`\ b_j && == v_j
\end{aligned}
$$

Thus, in our application, given a periodic function f, one can compute its Fourier series by taking the *innerF* product of it with each of $sin\ (n * x)\ /\ \sqrt{\pi}$ and $cos\ (n * x)\ /\ \sqrt{\pi}$.

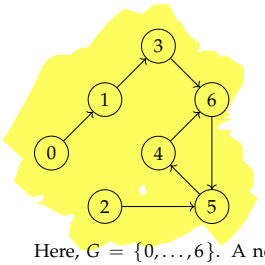
Exercise 7.7. Derive *derive* for this representation.

→ Exercise 7.7 very nice, perhaps show live ?

7.4.4 Simple deterministic systems (transition systems)

Simple deterministic systems are given by endofunctions on a finite set $next : G \to G$. They can often be conveniently represented as a graph, for example

[4] The proof can be easily adapted to infinite sums.



$G = \{0, \ldots, 6\}$

$next : G \to G$

Here, $G = \{0, \ldots, 6\}$. A node in the graph represents a state. A transition $i \to j$ means $next\ i = j$. Since *next* is an endofunction, every node must be the source of exactly one arrow.

We can take as vectors the characteristic functions of subsets of G, i.e., $G \to \{0, 1\}$. Now, $\{0, 1\}$ is not a field with respect to the standard arithmetical operations (it is not even closed with respect to addition), and the standard trick to workaround this issue is to extend the type of the functions to $\mathbb{R}$.

The canonical basis vectors are, as usual, $e\ i = V\ (is\ i)$. Each $e\ i$ is the characteristic function of a singleton set, $\{i\}$.

We can interpret $e\ (next\ 0), \ldots, e\ (next\ 6)$ as the images of the basis vectors $e\ 0, \ldots, e\ 6$ of $Vector\ \mathbb{R}\ G$ under the transformation

$$f :: Vector\ \mathbb{R}\ G \to Vector\ \mathbb{R}\ G$$
$$f\ (e\ i) = e\ (next\ i)$$

To write the matrix associated to f, we have to compute what vector is associated to each canonical basis vector:

$$M = \begin{bmatrix} f\ (e\ 0) & \cdots & f\ (e\ n) \end{bmatrix}$$

Therefore:

$$M = \begin{array}{c} \\ r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Notice that row 0 and row 2 contain only zero, as one would expect from the graph of *next*: no matter where we start from, the system will never reach node 0 or node 2.

Nicola.Botta@pik-potsdam.de

How do subsets of G behave under next ?

↳ Ex. Safe op. space !

Char. functions:

c char. func. of $A \subseteq G$

≡

$c : G \to Bool \wedge$

$c\ q = T \Leftrightarrow q \in A$

Char. functions are (sort of) vectors !

Vector S G ~ G → S

Starting with a canonical basis vector $e\ i$, we obtain $M * e\ i = f\ (e\ i)$, as we would expect. The more interesting thing is if we start with something different from a basis vector, say $[0,0,1,0,1,0,0] == e\ 2 + e\ 4$. We obtain $\{f\ 2, f\ 4\} = \{5,6\}$, the image of $\{2,4\}$ through f. In a sense, we can say that the two transitions happened in parallel. But that is not quite accurate: if we start with $\{3,4\}$, we no longer get the characteristic function of $\{f\ 3, f\ 4\} = \{6\}$, instead, we get a vector that does not represent a characteristic function at all: $[0,0,0,0,0,0,2] = 2 \lhd e\ 6$.

In general, if we start with an arbitrary vector, we can interpret this as starting with various quantities of some unspecified material in each state, simultaneously. If f were injective, the respective quantities would just get shifted around, but in our case, we get a more general behaviour.

What if we do want to obtain the characteristic function of the image of a subset? In that case, we need to use other operations than the standard arithmetical ones, for example *min* and *max*.

However, $(\{0,1\}, max, min)$ is not a field, and neither is $(\mathbb{R}, max, min)$. This means that we do not have a vector space, but rather a *module* (a generalisation of vector space). One can still do a lot with modules: for example the definition of matrix multiplication only demands a *Ring* rather than a *Field* (and none of the *VectorSpace* laws demand scalar division). Therefore, having just a module is not a problem if all we want is to compute the evolutions of possible states, but we cannot apply most of the deeper results of linear algebra.[5]

In the example above, we have:

```
newtype G = G Int deriving (Eq, Show)
instance Bounded G where minBound = G 0; maxBound = G 6
instance Enum G    where toEnum = G; fromEnum (G g) = g
```

Note that the *Ring* G instance is given just for convenient notation (integer literals): vector spaces in general do not rely on any numeric structure on the indices (G). The transition function has type $G \to G$ and the following implementation:

```
next1 :: G → G
next1 (G 0) = G 1;    next1 (G 1) = G 3;
next1 (G 2) = G 5;    next1 (G 3) = G 6;
next1 (G 4) = G 6;    next1 (G 5) = G 4;
next1 (G 6) = G 5
```

[5] For instance, such a deeper result would give ways to easily compute the stable states of a dynamic system.

Nicola.Botta@pik-potsdam.de

Take $S = \mathbb{R}$ and define

$f \circ e = ?$

The associated matrix is:

$M = Vars\ (f \circ e)$

$\longrightarrow$ Eve : m_1,

poor, !

...

Its associated matrix is

$$
\begin{array}{lll}
m\ g' & & \text{\{- } m \text{ is the matrix version of } f \text{ -\}} \\
V\ (\lambda g \to (f\ (e\ g)) & !\, g') & \text{\{- by the spec. of } f \text{ -\}} \\
V\ (\lambda g \to (e\ (next_1\ g)) & !\, g') & \text{\{- by def. of } e \text{ -\}} \\
V\ (\lambda g \to (V\ (is\ (next_1\ g))) & !\, g') & \text{\{- by def. of (!) -\}} \\
V\ (\lambda g \to is\ (next_1\ g)\ g') & & \text{\{- } is \text{ is symmetric -\}} \\
V\ (\lambda g \to is\ g'\ (next_1\ g)) & & \text{\{- by def. of } (\circ) \text{ -\}} \\
V\ (is\ g' \circ next_1) & &
\end{array}
$$

Thus we can implement m as:

$$
\begin{aligned}
&m_1 :: Ring\ s \Rightarrow G \to Vector\ s\ G \\
&m_1\ g' = V\ (is\ g' \circ next_1)
\end{aligned}
$$

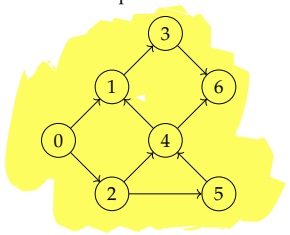
7.4.5 Non-deterministic systems

Another interpretation of the application of M to characteristic functions of a subset is the following: assuming that all I know is that the system is in one of the states of the subset, where can it end up after one step? (This assumes the *max-min* algebra as above.)

The general idea for non-deterministic systems, is that the result of applying the step function a number of times from a given starting state is a list of the possible states one could end up in.

In this case, the uncertainty is entirely caused by the fact that we do not know the exact initial state. However, there are cases in which the output of f is not known, even when the input is known. Such situations are modelled by endo-relations: $R : G \to G$, with $g\ R\ g'$ if g' is a potential successor of g. Endo-relations can also be pictured as graphs, but the restriction that every node should be the source of exactly one arrow is lifted. Every node can be the source of zero, one, or many arrows.

For example:



Nicola.Botta@pik-potsdam.de

How can we model such cases?

next : G -> ?

Now, starting in 0 we might end up either in 1 or 2 (but not both!). Starting in 6, the system breaks down: there is no successor state.

The matrix associated to R is built in the same fashion: we need to determine what vectors the canonical basis vectors are associated with:

$$M = \begin{array}{c} \\ r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

In Exercise 7.14 you are asked to start with $e\ 2 + e\ 3$ and iterate a number of times, to get a feeling for the possible evolutions.

Implementation The transition relation is given by:

```
f2 :: G → (G → Bool)
f2 (G 0) (G g) = g == 1 ∨ g == 2
f2 (G 1) (G g) = g == 3
f2 (G 2) (G g) = g == 4 ∨ g == 5
f2 (G 3) (G g) = g == 6
f2 (G 4) (G g) = g == 1 ∨ g == 6
f2 (G 5) (G g) = g == 4
f2 (G 6) (G g) = False
```

It has the associated matrix:

```
m2 g' = V (λg → f2 g g')
```

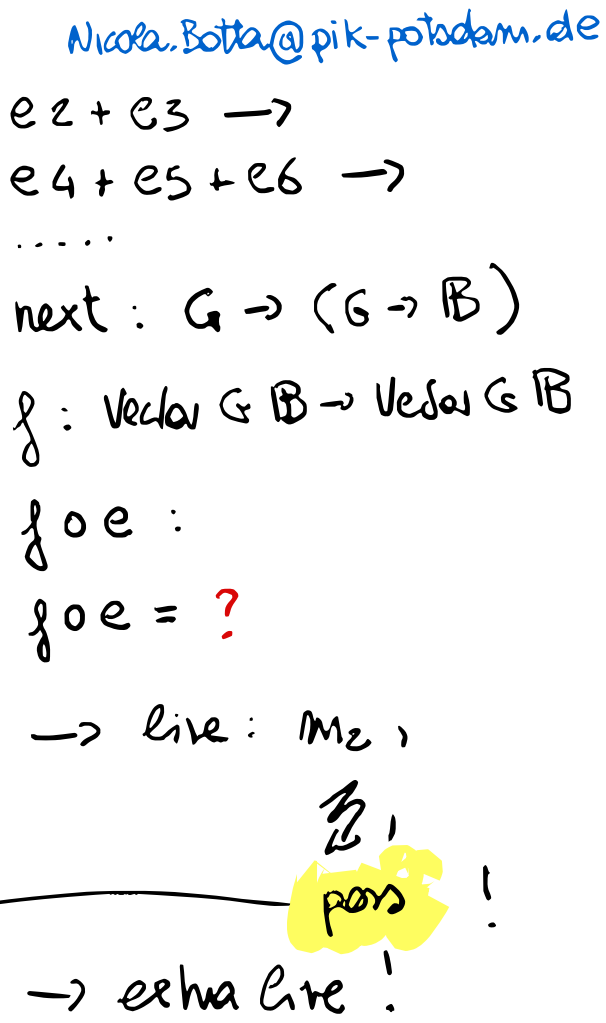
Even though *Bool* is not a *Field* (not even a *Ring*) the computations we need go through with these instances:

```
instance Additive       Bool where zero = False; (+) = (∨)
instance Multiplicative Bool where one  = True;  (*) = (∧)

instance AddGroup Bool where negate = error "negate: not used"
instance MulGroup Bool where recip  = id
```

As a test we compute the state after one step from "either 3 or 4":

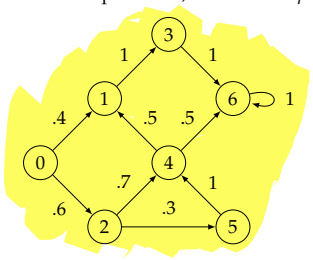
```
t2' = mulMV m2 (e (G 3) + e (G 4))
t2 = toL t2'   -- [False,True,False,False,False,False,True]
```



Nicola.Botta@pik-potsdam.de

7.4.6 Stochastic systems

Quite often, we have more information about the transition to possible future states. In particular, we can have *probabilities* of these transitions. For example



One could say that this case is a generalisation of the previous one, in which we can take all probabilities to be equally distributed among the various possibilities. While this is plausible, it is not entirely correct. For example, we have to introduce a transition from state 6 above. The nodes must be sources of *at least* one arrow.

In the case of the non-deterministic example, the "legitimate" inputs were characteristic functions, i.e., the "vector space" was $G \to \{0,1\}$ (the quotes are necessary because, as discussed, the target is not a field). In the case of stochastic systems, the inputs will be *probability distributions* over G, that is, functions $p : G \to [0,1]$ with the property that

$$sum\ [p\ g \mid g \leftarrow G] = 1$$

If we know the current probability distributions over states, then we can compute the next one by using the *total probability formula*, which can be expressed as

$$p\ a = sum\ [p\ (a \mid b) * p\ b \mid b \leftarrow G]$$

We study probability extensively in Chapter 9, but for now, let's just remark that the notation is extremely suspicious. The "argument" $(a \mid b)$, which is usually read "a, given b", is clearly not of the same type as a or b, so cannot really be an argument to p. Additionally, the $p\ a$ we are computing with this formula is not the $p\ a$ which must eventually appear in the products on the right hand side.

In any case, at this stage, what we need to know is that the conditional probability $p\ (a \mid b)$ gives us the probability that the next state is a, given that

why?

$next_1 : G \to G$

$next_2 : G \to (G \to B)$

$next_3 : G \to ?$

→ Exercise (extra)

Nicola.Botta@pik-potsdam.de

Exercise (exha): define a tp function that implements the lokal probability law

$$p\ a = sum\ [p(a|b) * p\ b \mid b \leftarrow G]$$

Hint:

```
tp : ?
tp p cp = ?
```

Which conditions have to fulfill p and cp for tp p cp to represent a probability dist.?

Nicola.Botta@pik-potsdam.de

the current state is b. But this is exactly the information summarised in the graphical representation. Moreover, it can be shown that the total probability formula is identical to a matrix-vector multiplication.

As usual, we write the associated matrix by looking at how the canonical basis vectors are transformed. In this case, the canonical basis vector $e\ i = \lambda j \to i\ `is`\ j$ is the probability distribution *concentrated* in i. This means that the probability to be in state i is 100% and the probability of being anywhere else is 0.

$$M = \begin{array}{c} \\ r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ .4 & 0 & 0 & 0 & .5 & 0 & 0 \\ .6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & .7 & 0 & 0 & 1 & 0 \\ 0 & 0 & .3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & .5 & 0 & 1 \end{pmatrix} \end{array}$$

As before, a good exercise is to explore the evolution of the system. For example, in Exercise 7.15 you are asked how many steps you need to take before the probability is concentrated in state 6 starting from state 0? And in Exercise 7.16 you are tasked with implementing the example by defining the transition function (giving the probability of getting to g' from g)

$$f_3 :: G \to \mathit{Vector}\ \mathbb{R}\ G$$

and the associated matrix

$$m_3 :: G \to \mathit{Vector}\ \mathbb{R}\ G$$

→ Exercise 7.15
→ Exercise 7.16
→ live !
→ Extra Exercise

7.4.7 *Quantum Systems

Instead of real numbers for probabilities, we could consider using complex numbers — one then speaks of "amplitudes". An amplitude represented by a complex number z is converted to a probability by taking the square of the modulus of z:

$$p\ z = \mathit{conj}\ z * z$$

We can then rewrite the law of total probability as follows:

$$\begin{aligned} & \mathit{sum}\ [p\ !\ i \mid i \leftarrow \mathit{finiteDomain}] \\ = {} & \mathit{sum}\ [\mathit{conj}\ (z\ !\ i) * (z\ !\ i) \mid i \leftarrow \mathit{finiteDomain}] \\ = {} & \mathit{inner}\ z\ z \end{aligned}$$

Indeed, for spaces with complex scalars, one should conjugate coefficients (of an orthonormal basis) when computing the inner products. Hence, rather conveniently, the law of total probability is replaced by conservation of the norm of state vectors. In particular, norms are conserved if the transition matrix is unitary.

The unitary character of the transition matrix defines valid systems from the point of view of quantum mechanics. Because all unitary matrices are invertible, it follows that all quantum mechanical systems have an invertible dynamics. Furthermore, the inverted matrix is also unitary, and therefore the inverted system is also valid as a quantum dynamical system.

Here is an example unitary matrix

$$M = \begin{array}{c} \\ r_0 \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{array} \begin{array}{c} \begin{array}{ccccccc} c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{2}/2 & -\sqrt{2}/2 & 0 & 0 \\ 0 & 0 & 0 & \sqrt{2}/2 & \sqrt{2}/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & \sqrt{3}/2 \\ 0 & 0 & 0 & 0 & 0 & \sqrt{3}/2 & -1/2 \end{pmatrix} \end{array}$$

In this example the amplitudes of states 0, 1, and 2 are permuted at every step. States 3 and 4 get mixed into one another, and one can note that the sign of their amplitudes may get inverted. A similar situation happens between states 5 and 6, but at a higher rate.

7.5 *Monadic dynamical systems

This section is meant to give perspective for the readers who are already familiar with monads. Even though it can be safely skipped, it presents a useful unified view of the previous sections which could help understanding the material.

All the examples of dynamical systems we have seen in the previous section have a similar structure. They work by taking a state (which is one of the generators) and return a structure of possible future states of type G:

- deterministic: there is exactly one possible future state: we take an element of G and return an element of G. The transition function has the type $f : G \to G$, the structure of the target is just G itself.

Nicola.Botta@pik-potsdam.de

Extra but we wrap-up:

- We have applied LA notions to model dynamical systems

- non-deterministic: there is a set of possible future states, which we have implemented as a characteristic function $G \to \{0,1\}$. The transition function has the type $f : G \to (G \to \{0,1\})$. The structure of the target is the *powerset* of G.
- stochastic: given a state, we compute a probability distribution over possible future states. The transition function has the type $f : G \to (G \to \mathbb{R})$, the structure of the target is the probability distributions over G.
- quantum: given an observable state, we compute a superposition of possible orthogonal future states.

Therefore:

- deterministic: $f : G \to Id\ G$
- non-deterministic: $f : G \to Powerset\ G$, where $Powerset\ G = G \to \{0,1\}$
- stochastic: $f : G \to Prob\ G$, where $Prob\ G = G \to [0,1]$
- quantum: $f : G \to Super\ G$, where $Super\ G = G \to Complex$. (Additionally f must be invertible)

We have represented the elements of the various structures as vectors. We also had a way of representing, as structures of possible states, those states that were known precisely: these were the canonical basis vectors $e\ i$. Due to the nature of matrix-vector multiplication, what we have done was in effect:

$$
\begin{aligned}
& M * v \quad \text{-- } v \text{ represents the current possible states} \\
= \ & \{\text{- } v \text{ is a linear combination of the basis vectors -}\} \\
& M * (v\ 0 \lhd e\ 0 + \ldots + v\ n \lhd e\ n) \\
= \ & \{\text{- homomorphism -}\} \\
& v\ 0 \lhd (M * e\ 0) + \ldots + v\ n \lhd (M * e\ n) \\
= \ & \{\text{- } e\ i \text{ represents the known current state } i\text{, therefore } M * e\ i = f\ i \text{ -}\} \\
& v\ 0 \lhd f\ 0 + \ldots + v\ n \lhd f\ n
\end{aligned}
$$

So, we apply f to every state, as if we were starting from precisely that state, obtaining the possible future states starting from that state, and then collect all these hypothetical possible future states in some way that takes into account the initial uncertainty (represented by $v\ 0$, ..., $v\ n$) and the nature of the uncertainty (the specific $(+)$ and $(\lhd)$).

If you examine the types of the operations involved

Nicola.Botta@pik-potsdam.de

- Deterministic

$X \to (X \to \mathbb{R})$

- Non-deterministic

$X \to (X \to \mathbb{B})$

- Stochastic

$X \to (X \to [0,1])$

If we generalize to

$X \to Poss\ X$

what we have done was

$$e : G \to \mathit{Possible}\ G$$

and

$$\mathit{flip}\ (*) : \mathit{Possible}\ G \to (G \to \mathit{Possible}\ G) \to \mathit{Possible}\ G$$

you see that they are very similar to the monadic operations

$$\mathit{return} : g \to m\ g$$
$$(\gg\!=) : m\ g \to (g \to m\ g') \to m\ g'$$

which suggests that the structure of possible future states might be monadic. Indeed, that is the case.

Since we implemented all these as matrix-vector multiplications, this raises the question: is there a monad underlying matrix-vector multiplication, such that the above are instances of it (obtained by specialising the scalar type S)? The answer is yes, up to a point, as we shall see in the next section.

Exercise 7.8 (*Hard). Write *Monad* instances for *Id*, *Powerset*, *Prob*, *Super*.

7.5.1 *The monad of linear algebra

Haskell *Monad*s, just like *Functor*s, require *return* and $\gg\!=$ to be defined for every type. This will not work, in general. Our definition will work for *finite types* only.

```
class FinFunc f where
  func :: (Finite a, Finite b) ⇒ (a → b) → f a → f b
class FinMon f where
  embed :: Finite a ⇒ a → f a
  bind  :: (Finite a, Finite b) ⇒ f a → (a → f b) → f b
```

The idea is that vectors on finite types are finite functors and monads:

```
instance Ring s ⇒ FinFunc (Vector s) where
  func f (V v) = V (λg' → sum [v g | g ← finiteDomain, g' == f g])
instance Field s ⇒ FinMon (Vector s) where
  embed = embedFinM
  bind  = bindFinM

embedFinM :: (Eq a, Ring s) ⇒ a → Vector s a
embedFinM g = V (is g)

bindFinM :: (Field s, Finite a) ⇒ Vector s a → (a → Vector s b) → Vector s b
bindFinM (V v) f = V (λg' → linComb v (λg → f g ! g'))
```

Nicola.Botta@pik-potsdam.de

1) define

next : X → Poss X

2) compute

f o e = V o next

M = mons f o e

3) Iterate M * starting from an initial lin. comb. of e vectors

→

Nicola.Botta@pik-potsdam.de

Types of the operations involved:

:t e ~ X → Poss X

:t flip (*) ~ Poss X → (X → Poss X) → Poss X

compare to the types of monadic operations:

:t ret = X → M X

:t bind = M X → (X → M Y) → M Y

Is Vector S a monad? —> Yes!

Nicola.Botta@pik-potsdam.de

Note that, if $v :: Vector\ S\ G$ and $f :: G \to Vector\ S\ G'$ then both $func\ f\ v$ and $bind\ v\ f$ are of type $Vector\ S\ G'$. How do these operations relate to linear algebra and matrix-vector multiplication?

Remember that $e\ g$ is that vector whose components are zero except for the gth one which is one. In other words

$$e\ g = V\ (is\ g) = embed\ g$$

and thus $embed = e$. In order to understand how matrix-vector multiplication relates to the monadic operations, remember that matrices are just functions of type $G \to Vector\ S\ G'$:

$$\textbf{type}\ Matrix\ s\ g\ g' = g' \to Vector\ s\ g$$

According to our earlier definition, we can rewrite matrix-vector multiplication in terms of *linComb*

$$\begin{aligned}
&mulMV\ m\ (V\ v)\\
=\ &\{\text{- earlier definition -}\}\\
&linComb\ v\ (transpose\ m)
\end{aligned}$$

Now we have:

$$\begin{aligned}
&mulMV\ (transpose\ m)\ (V\ v)\\
=\ &\{\text{- def. of } mulMV \text{ -}\}\\
&linComb\ v\ (transpose\ (transpose\ m))\\
=\ &\{\text{- Property of } transpose \text{ -}\}\\
&linComb\ v\ m\\
=\ &\{\text{- } linComb\text{-}V \text{ lemma -}\}\\
&V\ (\lambda i \to linComb\ v\ (\lambda j \to m\ j\ !\ i))\\
=\ &\{\text{- def. of } bind \text{ -}\}\\
&bind\ (V\ v)\ m
\end{aligned}$$

Thus we see that $bind\ v\ f$ is "just" a matrix-vector multiplication.

The *linComb-V* lemma says that for $a :: j \to s$ and $v :: j \to Vector\ s\ i$ we have $linComb\ a\ v == V\ (\lambda i \to linComb\ a\ (\lambda j \to v\ j\ !\ i))$. The proof uses the definitions of *linComb*, $(\lhd)$, and the *Additive* instances for *Vector* and functions but is omitted here for brevity.

7.6 Associated code

Conversions and *Show* functions so that we can actually see our vectors.

Nicola.Botta@pik-potsdam.de

```
toL :: Finite g ⇒ Vector s g → [s]
toL (V v) = map v finiteDomain

instance (Finite g, Show s) ⇒ Show (g → s)      where show = showFun
instance (Finite g, Show s) ⇒ Show (Vector s g) where show = showVec

showVec :: (Finite g, Show s) ⇒ Vector s g → String
showVec (V v) = showFun v

showFun :: (Finite a, Show b) ⇒ (a → b) → String
showFun f = show (map f finiteDomain)
```

Nicola.Botta@pik-potsdam.de

7.7 Exercises

Some exercises were inlined in the chapter text, and here are a few more.

Exercise 7.9. Compute $((M*) \circ e)\ g\ g'$.

Exercise 7.10. Matrix-matrix multiplication is defined in order to ensure a homomorphism from $(*)$ to $(\circ)$.

$$\forall\, M.\ \forall\, M'.\ ((M' * M)*) == (M'*) \circ (M*)$$

or in other words

$$H_2((*), (*), (\circ))$$

Work out the types and expand the definitions to verify that this claim holds. Note that one $(*)$ is matrix-vector multiplication and the other is matrix-matrix multiplication.

Exercise 7.11. Show that matrix-matrix multiplication is associative.

Exercise 7.12. With $G = \mathbb{N}$ for the set of indices, write the infinite-dimensional matrix representing D for power series.

Exercise 7.13. Write the matrix I_n associated with integration of polynomials of degree n.

Exercise 7.14. In the context of Section 7.4.5: start with $v_0 = e\ 2 + e\ 3$ and iterate $M*$ a number of times, to get a feeling for the possible evolutions. What do you notice? What is the largest number of steps you can make before the result is the origin vector (just zero)?

Now change M to M' by inverting the arrow from 2 to 4 and repeat the exercise. What changes? Can you prove it?

Exercise 7.15. In the context of the example matrix M in Section 7.4.6: starting from state 0, how many steps do you need to take before the probability is concentrated in state 6?

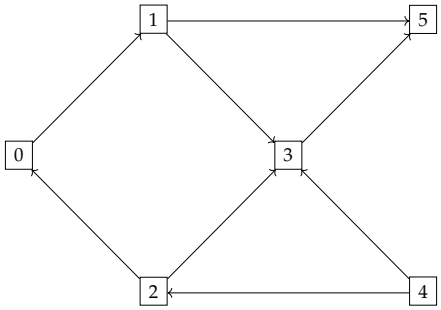
Reverse again the arrow from 2 to 4 (so that $2 \to 5$ has probability 1, $4 \to 2$ probability 0.7, $4 \to 6$ and $4 \to 1$ have probability 0.15 each). What can you say about the long-term behaviour of the system now?

Exercise 7.16. In the context of the example matrix M in Section 7.4.6: implement the example. You will need to define the transition function of type $G \to (G \to [0,1])$ returning the probability of getting from g to g', and the associated matrix.

Nicola.Botta@pik-potsdam.de

Exercise 7.17. *From exam 2017-03-14*

Consider a non-deterministic system with a transition function $f : G \to [G]$ (for $G = \{0\,..\,5\}$) represented in the following graph



The transition matrix can be given the type $m :: G \to (G \to \mathit{Bool})$ and the canonical vectors have type $e\,i :: G \to \mathit{Bool}$ for i in G.

1. (General questions.) What do the canonical vectors represent? What about non-canonical ones? What are the operations on *Bool* used in the matrix-vector multiplication?

2. (Specific questions.) Write the transition matrix m of the system. Compute, using matrix-vector multiplication, the result of three steps of the system starting in state 2.