Category theory
as an abstract programming language

Mohamed Barakat

Universität Siegen

Journées Nationales de Calcul Formel 2019
CIRM, Luminy
4–8 février 2019

Joint work with
Markus Lange-Hegermann, Sebastian Gutsche, Florian Heiderich,
Sebastian Posur, Kamal Saleh

Intersection: From concrete algorithms to abstraction

Example (Intersection of subspaces)

Compute the intersection of the two subspaces of $V \coloneqq \mathbb{Q}^{3\times 1}$

$$U_1 \coloneqq \langle (1,2,3),(2,3,4),(0,1,2) \rangle,$$
$$U_2 \coloneqq \langle (1,2,4),(3,2,0) \rangle.$$

Intersection: From concrete algorithms to abstraction

Example (Intersection of subspaces)

Compute the intersection of the two subspaces of $V := \mathbb{Q}^{3\times 1}$

$$U_1 := \langle (1,2,3), (2,3,4), (0,1,2) \rangle,$$
$$U_2 := \langle (1,2,4), (3,2,0) \rangle.$$

Solution:

$$U_1 \cap U_2 = \langle (1,1,1) \rangle < \mathbb{Q}^{3\times 1}.$$

Intersection: From concrete algorithms to abstraction

Example (Intersection of subspaces)

Compute the intersection of the two subspaces of $V \coloneqq \mathbb{Q}^{3\times 1}$

$$U_1 \coloneqq \langle (1,2,3),(2,3,4),(0,1,2) \rangle,$$
$$U_2 \coloneqq \langle (1,2,4),(3,2,0) \rangle.$$

Solution:

$$U_1 \cap U_2 = \langle (1,1,1) \rangle < \mathbb{Q}^{3\times 1}.$$

Goals

- Describe algorithms to intersect vector subspaces;

Intersection: From concrete algorithms to abstraction

Example (Intersection of subspaces)

Compute the intersection of the two subspaces of $V := \mathbb{Q}^{3\times 1}$

$$U_1 := \langle (1,2,3), (2,3,4), (0,1,2) \rangle,$$
$$U_2 := \langle (1,2,4), (3,2,0) \rangle.$$

Solution:

$$U_1 \cap U_2 = \langle (1,1,1) \rangle < \mathbb{Q}^{3\times 1}.$$

Goals

- Describe algorithms to intersect vector subspaces;
- Generalize these algorithms to more general setups.

Intersection: From concrete algorithms to abstraction

Example (Intersection of subspaces)

Compute the intersection of the two subspaces of $V := \mathbb{Q}^{3\times 1}$

$$U_1 := \langle (1,2,3), (2,3,4), (0,1,2) \rangle,$$
$$U_2 := \langle (1,2,4), (3,2,0) \rangle.$$

Solution:

$$U_1 \cap U_2 = \langle (1,1,1) \rangle < \mathbb{Q}^{3\times 1}.$$

Goals

- Describe algorithms to intersect vector subspaces;
- Generalize these algorithms to more general setups.

Algorithm 1 to intersect two vector subspaces

Algorithm 1: Intersection of vector subspaces

Algorithm 1 to intersect two vector subspaces

Algorithm 1: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$

Algorithm 1 to intersect two vector subspaces

Algorithm 1: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$

$U_1 := \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 := \langle$rows of the matrix $\mathtt{u}_2\rangle$.

Algorithm 1 to intersect two vector subspaces

Algorithm 1: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 := \langle \text{rows of the matrix } \mathtt{u}_1 \rangle, U_2 := \langle \text{rows of the matrix } \mathtt{u}_2 \rangle$.
Output: $\mathtt{i}_1$ with $U_1 \cap U_2 = \langle \text{rows of the matrix } \mathtt{i}_1 \rangle \leq \mathbb{Q}^{1 \times d}$

Algorithm 1 to intersect two vector subspaces

Algorithm 1: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 \coloneqq \langle \text{rows of the matrix } \mathtt{u}_1 \rangle, U_2 \coloneqq \langle \text{rows of the matrix } \mathtt{u}_2 \rangle$.
Output: $\mathtt{i}_1$ with $U_1 \cap U_2 = \langle \text{rows of the matrix } \mathtt{i}_1 \rangle \leq \mathbb{Q}^{1 \times d}$
Intersection1 $(\mathtt{u}_1, \mathtt{u}_2)$

Algorithm 1 to intersect two vector subspaces

Algorithm 1: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$

$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.

Output: $\mathtt{i}_1$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{i}_1\rangle \leq \mathbb{Q}^{1 \times d}$

Intersection1 *($\mathtt{u}_1, \mathtt{u}_2$)*

```
m₁ := REF(u₁)                    // row echelon form of u₁
```

Algorithm 1: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.
Output: $\mathtt{i}_1$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{i}_1\rangle \leq \mathbb{Q}^{1 \times d}$

Intersection1 *($\mathtt{u}_1, \mathtt{u}_2$)*

```
m₁ := REF(u₁)        // row echelon form of u₁
m₂ := REF(u₂)
```

Algorithm 1: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{?\times d}$

$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.

Output: $\mathtt{i}_1$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{i}_1\rangle \leq \mathbb{Q}^{1\times d}$

Intersection1 *($\mathtt{u}_1, \mathtt{u}_2$)*

1 $\mathtt{m}_1 \coloneqq \mathtt{REF}(\mathtt{u}_1)$ // row echelon form of $\mathbf{u}_1$

2 $\mathtt{m}_2 \coloneqq \mathtt{REF}(\mathtt{u}_2)$

3 $\left(\begin{array}{c|c} \mathtt{n}_1 & \mathtt{n}_2 \end{array}\right) \coloneqq \mathtt{LeftNullSpace}\left(\left(\begin{array}{c} \mathtt{m}_1 \\ \hline \mathtt{m}_2 \end{array}\right)\right)$

Algorithm 1 to intersect two vector subspaces

Algorithm 1: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$

$U_1 := \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 := \langle$rows of the matrix $\mathtt{u}_2\rangle$.

Output: $\mathtt{i}_1$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{i}_1\rangle \leq \mathbb{Q}^{1 \times d}$

Intersection1 *($\mathtt{u}_1, \mathtt{u}_2$)*

1 $\mathtt{m}_1 := \mathtt{REF}(\mathtt{u}_1)$ // row echelon form of $\mathbf{u}_1$

2 $\mathtt{m}_2 := \mathtt{REF}(\mathtt{u}_2)$

3 $\left(\begin{array}{c|c} \mathtt{n}_1 & \mathtt{n}_2 \end{array} \right) := \mathtt{LeftNullSpace}\left(\begin{pmatrix} \mathtt{m}_1 \\ \hline \mathtt{m}_2 \end{pmatrix}\right)$

4 $\mathtt{i}_1 := \mathtt{MatMul}(\mathtt{n}_1, \mathtt{m}_1) := \mathtt{n}_1 \mathtt{m}_1$

Algorithm 1 to intersect two vector subspaces

Algorithm 1: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$

$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.

Output: $\mathtt{i}_1$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{i}_1\rangle \leq \mathbb{Q}^{1 \times d}$

Intersection1 *($\mathtt{u}_1, \mathtt{u}_2$)*

$\mathtt{m}_1 \coloneqq \mathtt{REF}(\mathtt{u}_1)$ // row echelon form of $\mathbf{u}_1$

$\mathtt{m}_2 \coloneqq \mathtt{REF}(\mathtt{u}_2)$

$\left(\; \mathtt{n}_1 \; \middle| \; \mathtt{n}_2 \; \right) \coloneqq \mathtt{LeftNullSpace}\left(\begin{pmatrix} \mathtt{m}_1 \\ \hline \mathtt{m}_2 \end{pmatrix}\right)$

$\mathtt{i}_1 \coloneqq \mathtt{MatMul}(\mathtt{n}_1, \mathtt{m}_1) \coloneqq \mathtt{n}_1 \mathtt{m}_1$

return $\mathtt{i}_1$

Algorithm 2: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.
Output: $\mathtt{s}_1$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{s}_1\rangle \leq \mathbb{Q}^{1 \times d}$
Intersection2 *($\mathtt{u}_1, \mathtt{u}_2$)*

Algorithm 2 to intersect two vector subspaces

Algorithm 2: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 := \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 := \langle$rows of the matrix $\mathtt{u}_2\rangle$.
Output: $\mathtt{s}_1$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{s}_1\rangle \leq \mathbb{Q}^{1 \times d}$
Intersection2 *($\mathtt{u}_1, \mathtt{u}_2$)*

```
1    e₂ := RightNullSpace(u₂)
```

Algorithm 2: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 := \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 := \langle$rows of the matrix $\mathtt{u}_2\rangle$.

Output: $\mathtt{s}_1$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{s}_1\rangle \leq \mathbb{Q}^{1 \times d}$

Intersection2 *($\mathtt{u}_1, \mathtt{u}_2$)*

```
e₂ := RightNullSpace(u₂)
w₁ := MatMul(u₁, e₂) := u₁e₂
```

Algorithm 2: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.
Output: $\mathtt{s}_1$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{s}_1\rangle \leq \mathbb{Q}^{1 \times d}$
Intersection2 *($\mathtt{u}_1, \mathtt{u}_2$)*

1. $\mathtt{e}_2 \coloneqq \mathtt{RightNullSpace}(\mathtt{u}_2)$
2. $\mathtt{w}_1 \coloneqq \mathtt{MatMul}(\mathtt{u}_1, \mathtt{e}_2) \coloneqq \mathtt{u}_1\mathtt{e}_2$
3. $\mathtt{k}_1 \coloneqq \mathtt{LeftNullSpace}(\mathtt{w}_1)$

Algorithm 2 to intersect two vector subspaces

Algorithm 2: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$

$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.

Output: $\mathtt{s}_1$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{s}_1\rangle \leq \mathbb{Q}^{1 \times d}$

Intersection2 *($\mathtt{u}_1, \mathtt{u}_2$)*

```
e₂ := RightNullSpace(u₂)
w₁ := MatMul(u₁, e₂) := u₁e₂
k₁ := LeftNullSpace(w₁)
v₁ := MatMul(k₁, u₁) := k₁u₁
```

Algorithm 2 to intersect two vector subspaces

Algorithm 2: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$

$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.

Output: $\mathtt{s}_1$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{s}_1\rangle \leq \mathbb{Q}^{1 \times d}$

Intersection2 *(*$\mathtt{u}_1, \mathtt{u}_2$*)*

$\mathtt{e}_2 \coloneqq \mathtt{RightNullSpace}(\mathtt{u}_2)$

$\mathtt{w}_1 \coloneqq \mathtt{MatMul}(\mathtt{u}_1, \mathtt{e}_2) \coloneqq \mathtt{u}_1 \mathtt{e}_2$

$\mathtt{k}_1 \coloneqq \mathtt{LeftNullSpace}(\mathtt{w}_1)$

$\mathtt{v}_1 \coloneqq \mathtt{MatMul}(\mathtt{k}_1, \mathtt{u}_1) \coloneqq \mathtt{k}_1 \mathtt{u}_1$

$\mathtt{s}_1 \coloneqq \mathtt{REF}(\mathtt{v}_1)$

Algorithm 2 to intersect two vector subspaces

Algorithm 2: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.

Output: $\mathtt{s}_1$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{s}_1\rangle \leq \mathbb{Q}^{1 \times d}$

Intersection2 *($\mathtt{u}_1, \mathtt{u}_2$)*

$\mathtt{e}_2 \coloneqq \mathtt{RightNullSpace}(\mathtt{u}_2)$
$\mathtt{w}_1 \coloneqq \mathtt{MatMul}(\mathtt{u}_1, \mathtt{e}_2) \coloneqq \mathtt{u}_1 \mathtt{e}_2$
$\mathtt{k}_1 \coloneqq \mathtt{LeftNullSpace}(\mathtt{w}_1)$
$\mathtt{v}_1 \coloneqq \mathtt{MatMul}(\mathtt{k}_1, \mathtt{u}_1) \coloneqq \mathtt{k}_1 \mathtt{u}_1$
$\mathtt{s}_1 \coloneqq \mathtt{REF}(\mathtt{v}_1)$
return $\mathtt{s}_1$

Algorithm 3 to intersect two vector subspaces

Algorithm 3: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.
Output: $\mathtt{k}$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{k}\rangle \leq \mathbb{Q}^{1 \times d}$
Intersection3 *($\mathtt{u}_1, \mathtt{u}_2$)*

Algorithm 3 to intersect two vector subspaces

Algorithm 3: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 := \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 := \langle$rows of the matrix $\mathtt{u}_2\rangle$.
Output: $\mathtt{k}$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{k}\rangle \leq \mathbb{Q}^{1 \times d}$
Intersection3 *(*$\mathtt{u}_1, \mathtt{u}_2$*)*

```
e₁ := RightNullSpace(u₁)
```

Algorithm 3 to intersect two vector subspaces

Algorithm 3: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{?\times d}$

$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.

Output: $\mathtt{k}$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{k}\rangle \leq \mathbb{Q}^{1\times d}$

Intersection3 *($\mathtt{u}_1, \mathtt{u}_2$)*

```
e1 := RightNullSpace(u1)
e2 := RightNullSpace(u2)
```

Algorithm 3 to intersect two vector subspaces

Algorithm 3: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.
Output: $\mathtt{k}$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{k}\rangle \leq \mathbb{Q}^{1 \times d}$

Intersection3 *($\mathtt{u}_1, \mathtt{u}_2$)*

```
e1 := RightNullSpace(u1)
e2 := RightNullSpace(u2)
a := Augment(e1, e2)
```

Algorithm 3 to intersect two vector subspaces

Algorithm 3: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.
Output: $\mathtt{k}$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{k}\rangle \leq \mathbb{Q}^{1 \times d}$

Intersection3 *($\mathtt{u}_1, \mathtt{u}_2$)*

$\mathtt{e}_1 \coloneqq$ RightNullSpace$(\mathtt{u}_1)$
$\mathtt{e}_2 \coloneqq$ RightNullSpace$(\mathtt{u}_2)$
$\mathtt{a} \coloneqq$ Augment$(\mathtt{e}_1, \mathtt{e}_2)$
$\mathtt{k} \coloneqq$ LeftNullSpace$(\mathtt{a})$

Algorithm 3 to intersect two vector subspaces

Algorithm 3: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.
Output: $\mathtt{k}$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{k}\rangle \leq \mathbb{Q}^{1 \times d}$

```
Intersection3 (u1, u2)
    e1 := RightNullSpace(u1)
    e2 := RightNullSpace(u2)
    a := Augment(e1, e2)
    k := LeftNullSpace(a)
    return k
```

Algorithm 4 = 3' to intersect two vector subspaces

Algorithm 4: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 := \langle \text{rows of the matrix } \mathtt{u}_1 \rangle, U_2 := \langle \text{rows of the matrix } \mathtt{u}_2 \rangle$.
Output: $\mathtt{z}_0$ with $U_1 \cap U_2 = \langle \text{rows of the matrix } \mathtt{z}_0 \rangle \leq \mathbb{Q}^{1 \times d}$
Intersection4 *($\mathtt{u}_1, \mathtt{u}_2$)*

Algorithm 4: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.
Output: $\mathtt{z}_0$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{z}_0\rangle \leq \mathbb{Q}^{1 \times d}$

Intersection4 $(\mathtt{u}_1, \mathtt{u}_2)$

```
d := NrColumns(u1)
```

Algorithm 4 = 3' to intersect two vector subspaces

Algorithm 4: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.
Output: $\mathtt{z}_0$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{z}_0\rangle \leq \mathbb{Q}^{1 \times d}$

Intersection4 $(\mathtt{u}_1, \mathtt{u}_2)$

```
d := NrColumns(u1)
i := IdentityMat(d, Q)
```

Algorithm 4: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{?\times d}$

$U_1 \coloneqq \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 \coloneqq \langle$rows of the matrix $\mathtt{u}_2\rangle$.

Output: $\mathtt{z}_0$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{z}_0\rangle \leq \mathbb{Q}^{1\times d}$

Intersection4 *($\mathtt{u}_1, \mathtt{u}_2$)*

$\mathtt{d} \coloneqq \texttt{NrColumns}(\mathtt{u}_1)$

$\mathtt{i} \coloneqq \texttt{IdentityMat}(\mathtt{d}, \mathbb{Q})$

$$\mathtt{p} \coloneqq \texttt{Stack}(\texttt{Augment}(\mathtt{i},\mathtt{i}), \texttt{Diag}(\mathtt{u}_1,\mathtt{u}_2)) \coloneqq \begin{pmatrix} 1 & 1 \\ \hline \mathtt{u}_1 & 0 \\ 0 & \mathtt{u}_2 \end{pmatrix}$$

Algorithm 4 = 3' to intersect two vector subspaces

Algorithm 4: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{?\times d}$
$U_1 \coloneqq \langle \text{rows of the matrix } \mathtt{u}_1 \rangle, U_2 \coloneqq \langle \text{rows of the matrix } \mathtt{u}_2 \rangle$.
Output: $\mathtt{z}_0$ with $U_1 \cap U_2 = \langle \text{rows of the matrix } \mathtt{z}_0 \rangle \leq \mathbb{Q}^{1\times d}$

Intersection4 *($\mathtt{u}_1, \mathtt{u}_2$)*

$\mathtt{d} \coloneqq \mathtt{NrColumns}(\mathtt{u}_1)$

$\mathtt{i} \coloneqq \mathtt{IdentityMat}(\mathtt{d}, \mathbb{Q})$

$$\mathtt{p} \coloneqq \mathtt{Stack}(\mathtt{Augment}(\mathtt{i}, \mathtt{i}), \mathtt{Diag}(\mathtt{u}_1, \mathtt{u}_2)) \coloneqq \left(\begin{array}{cc} 1 & 1 \\ \hline \mathtt{u}_1 & 0 \\ 0 & \mathtt{u}_2 \end{array}\right)$$

$$\left(\begin{array}{c|cc} \mathtt{z}_0 & \mathtt{z}_1 & \mathtt{z}_2 \end{array} \right) \coloneqq \mathtt{LeftNullSpace}(\mathtt{p})$$

Algorithm 4 = 3' to intersect two vector subspaces

Algorithm 4: Intersection of vector subspaces

Input: Two stackable matrices $\mathtt{u}_1, \mathtt{u}_2 \in \mathbb{Q}^{? \times d}$
$U_1 := \langle$rows of the matrix $\mathtt{u}_1\rangle, U_2 := \langle$rows of the matrix $\mathtt{u}_2\rangle$.
Output: $\mathtt{z}_0$ with $U_1 \cap U_2 = \langle$rows of the matrix $\mathtt{z}_0\rangle \leq \mathbb{Q}^{1 \times d}$
Intersection4 *($\mathtt{u}_1, \mathtt{u}_2$)*

$\mathtt{d} := \mathtt{NrColumns}(\mathtt{u}_1)$

$\mathtt{i} := \mathtt{IdentityMat}(\mathtt{d}, \mathbb{Q})$

$$\mathtt{p} := \mathtt{Stack}(\mathtt{Augment}(\mathtt{i}, \mathtt{i}), \mathtt{Diag}(\mathtt{u}_1, \mathtt{u}_2)) := \left(\begin{array}{cc} 1 & 1 \\ \hline \mathtt{u}_1 & 0 \\ 0 & \mathtt{u}_2 \end{array}\right)$$

$\left(\begin{array}{c|cc} \mathtt{z}_0 & \mathtt{z}_1 & \mathtt{z}_2 \end{array}\right) := \mathtt{LeftNullSpace}(\mathtt{p})$

return $\mathtt{z}_0$

Our goals were ...

Goals

- Describe algorithms to intersect vector subspaces;
- Generalize these algorithms to more general setups.

Our goals were ...

Goals

- Describe algorithms to intersect vector subspaces;
- Generalize these algorithms to more general setups.

Our goals were ...

Goals

- Describe algorithms to intersect vector subspaces;
- Generalize these algorithms to more general setups.

Main idea

Describe the subspaces $U_1, U_2 \leq V$ as the image of linear maps u_1, u_2 defined by the matrices $\mathtt{u}_1, \mathtt{u}_2$, respectively:

$$u_1 : \mathbb{Q}^{g_1 \times 1} \xrightarrow{\mathtt{u}_1} \mathbb{Q}^{d \times 1},$$
$$u_2 : \mathbb{Q}^{g_2 \times 1} \xrightarrow{\mathtt{u}_2} \mathbb{Q}^{d \times 1}.$$

Our goals were ...

Goals

- Describe algorithms to intersect vector subspaces;
- Generalize these algorithms to more general setups.

Main idea

Describe the subspaces $U_1, U_2 \leq V$ as the image of linear maps u_1, u_2 defined by the matrices $\mathtt{u}_1, \mathtt{u}_2$, respectively:

$$u_1 : \mathbb{Q}^{g_1 \times 1} \xrightarrow{\mathtt{u}_1} \mathbb{Q}^{d \times 1},$$
$$u_2 : \mathbb{Q}^{g_2 \times 1} \xrightarrow{\mathtt{u}_2} \mathbb{Q}^{d \times 1}.$$

Vector spaces together with their linear maps form a **category**.

Data structures and algorithms for a category

A **quiver** (directed multi-graph) $\mathcal{C}$ consists of

Data structures and algorithms for a category

A **quiver** (directed multi-graph) $\mathcal{C}$ consists of

- a class of objects $\mathcal{C}_0$;

Data structures and algorithms for a category

A **quiver** (directed multi-graph) $\mathcal{C}$ consists of

- a class of objects $\mathcal{C}_0$;
- a class of morphisms $\mathcal{C}_1$

Data structures and algorithms for a category

A **quiver** (directed multi-graph) $\mathcal{C}$ consists of

- a class of objects $\mathcal{C}_0$;
- a class of morphisms $\mathcal{C}_1$

- two structure maps:

(1,2) source and target $s, t : \mathcal{C}_1 \to \mathcal{C}_0$;

Data structures and algorithms for a category

A **quiver** (directed multi-graph) $\mathcal{C}$ consists of

- a class of objects $\mathcal{C}_0$;
- a class of morphisms $\mathcal{C}_1 := \dot{\bigcup}_{M,N\in\mathcal{C}_0} \underbrace{\operatorname{Hom}_{\mathcal{C}}(M,N)}_{(s\times t)^{-1}(M,N)}$;
- two structure maps:

(1,2) source and target $s, t : \mathcal{C}_1 \to \mathcal{C}_0$;

Data structures and algorithms for a category

A **quiver** (directed multi-graph) $\mathcal{C}$ consists of

- a class of objects $\mathcal{C}_0$;
- a class of morphisms $\mathcal{C}_1 := \dot{\bigcup}_{M,N \in \mathcal{C}_0} \underbrace{\operatorname{Hom}_{\mathcal{C}}(M,N)}_{(s \times t)^{-1}(M,N)}$;
- two structure maps:

(1,2) source and target $s, t : \mathcal{C}_1 \to \mathcal{C}_0$;

A **category** is a quiver $\mathcal{C}$ with two further structure maps:

Data structures and algorithms for a category

A **quiver** (directed multi-graph) $\mathcal{C}$ consists of

- a class of objects $\mathcal{C}_0$;
- a class of morphisms $\mathcal{C}_1 := \dot{\bigcup}_{M,N \in \mathcal{C}_0} \underbrace{\mathrm{Hom}_{\mathcal{C}}(M,N)}_{(s \times t)^{-1}(M,N)}$;
- two structure maps:

(1,2) source and target $s, t : \mathcal{C}_1 \to \mathcal{C}_0$;

A **category** is a quiver $\mathcal{C}$ with two further structure maps:

(3) the identity $1 : \mathcal{C}_0 \to \mathcal{C}_1$;

Data structures and algorithms for a category

A **quiver** (directed multi-graph) $\mathcal{C}$ consists of

- a class of objects $\mathcal{C}_0$;
- a class of morphisms $\mathcal{C}_1 := \dot{\bigcup}_{M,N\in\mathcal{C}_0} \underbrace{\mathrm{Hom}_{\mathcal{C}}(M,N)}_{(s\times t)^{-1}(M,N)}$;
- two structure maps:

(1,2) source and target $s, t : \mathcal{C}_1 \to \mathcal{C}_0$;

A **category** is a quiver $\mathcal{C}$ with two further structure maps:

(3) the identity $1 : \mathcal{C}_0 \to \mathcal{C}_1$;

(4) the “composition” $\mu : \mathcal{C}_1 \times_{\mathcal{C}_0} \mathcal{C}_1 \to \mathcal{C}_1, (\varphi, \psi) \mapsto \varphi\psi$

Data structures and algorithms for a category

A **quiver** (directed multi-graph) $\mathcal{C}$ consists of

- a class of objects $\mathcal{C}_0$;
- a class of morphisms $\mathcal{C}_1 := \dot{\bigcup}_{M,N \in \mathcal{C}_0} \underbrace{\mathrm{Hom}_{\mathcal{C}}(M,N)}_{(s \times t)^{-1}(M,N)}$;
- two structure maps:

(1,2) source and target $s, t : \mathcal{C}_1 \to \mathcal{C}_0$;

A **category** is a quiver $\mathcal{C}$ with two further structure maps:

(3) the identity $1 : \mathcal{C}_0 \to \mathcal{C}_1$;

(4) the "composition" $\mu : \mathcal{C}_1 \times_{\mathcal{C}_0} \mathcal{C}_1 \to \mathcal{C}_1, (\varphi, \psi) \mapsto \varphi\psi$

subject to the obvious relations.

Data structures and algorithms for a category

A **quiver** (directed multi-graph) $\mathcal{C}$ consists of

- a class of objects $\mathcal{C}_0$;
- a class of morphisms $\mathcal{C}_1 := \dot{\bigcup}_{M,N \in \mathcal{C}_0} \underbrace{\mathrm{Hom}_{\mathcal{C}}(M,N)}_{(s \times t)^{-1}(M,N)}$;
- two structure maps:

(1,2) source and target $s, t : \mathcal{C}_1 \to \mathcal{C}_0$;

A **category** is a quiver $\mathcal{C}$ with two further structure maps:

(3) the identity $1 : \mathcal{C}_0 \to \mathcal{C}_1$;

(4) the "composition" $\mu : \mathcal{C}_1 \times_{\mathcal{C}_0} \mathcal{C}_1 \to \mathcal{C}_1, (\varphi, \psi) \mapsto \varphi\psi$

subject to the obvious relations.

To describe an *instance* of a category we need two data structures (for $\mathcal{C}_0, \mathcal{C}_1$) and four algorithms (for $s, t, 1, \mu$).

Data structures and algorithms for a category

A **quiver** (directed multi-graph) $\mathcal{C}$ consists of

- a class of objects $\mathcal{C}_0$;
- a class of morphisms $\mathcal{C}_1 := \dot{\bigcup}_{M,N\in\mathcal{C}_0} \underbrace{\mathrm{Hom}_{\mathcal{C}}(M,N)}_{(s\times t)^{-1}(M,N)}$;
- two structure maps:

(1,2) source and target $s,t : \mathcal{C}_1 \to \mathcal{C}_0$;

A **category** is a quiver $\mathcal{C}$ with two further structure maps:

(3) the identity $1 : \mathcal{C}_0 \to \mathcal{C}_1$;

(4) the "composition" $\mu : \mathcal{C}_1 \times_{\mathcal{C}_0} \mathcal{C}_1 \to \mathcal{C}_1, (\varphi,\psi) \mapsto \varphi\psi$

subject to the obvious relations.

To describe an *instance* of a category we need two data structures (for $\mathcal{C}_0, \mathcal{C}_1$) and four algorithms (for $s, t, 1, \mu$).

Categories up to equivalence emphasize morphisms and treat objects merely as place holders for sources and targets.

Further definitions and examples

Definition

A category is called

- **small** if both $\mathcal{C}_0$ and $\mathcal{C}_1$ are sets;

Definition

A category is called

- **small** if both $\mathcal{C}_0$ and $\mathcal{C}_1$ are sets;
- **thin** if between two objects there is at most one morphism;

Further definitions and examples

Definition

A category is called

- **small** if both $\mathcal{C}_0$ and $\mathcal{C}_1$ are sets;
- **thin** if between two objects there is at most one morphism;
- **skeletal** if isomorphic objects are equal.

Further definitions and examples

Definition

A category is called

- **small** if both $\mathcal{C}_0$ and $\mathcal{C}_1$ are sets;
- **thin** if between two objects there is at most one morphism;
- **skeletal** if isomorphic objects are equal.

Example (Categories generalize classical notions)

- monoid $\equiv$ small category on one object

Further definitions and examples

Definition

A category is called

- **small** if both $\mathcal{C}_0$ and $\mathcal{C}_1$ are sets;
- **thin** if between two objects there is at most one morphism;
- **skeletal** if isomorphic objects are equal.

Example (Categories generalize classical notions)

- monoid $\equiv$ small category on one object
  A category is a “monoid on many objects”.

Further definitions and examples

Definition

A category is called

- **small** if both $\mathcal{C}_0$ and $\mathcal{C}_1$ are sets;
- **thin** if between two objects there is at most one morphism;
- **skeletal** if isomorphic objects are equal.

Example (Categories generalize classical notions)

- monoid ≡ small category on one object
  A category is a "monoid on many objects".
- pre-ordered set (proset) ≡ thin small category

Definition

A category is called

- **small** if both $\mathcal{C}_0$ and $\mathcal{C}_1$ are sets;
- **thin** if between two objects there is at most one morphism;
- **skeletal** if isomorphic objects are equal.

Example (Categories generalize classical notions)

- monoid ≡ small category on one object
  A category is a "monoid on many objects".
- pre-ordered set (proset) ≡ thin small category
- partially ordered set (poset) ≡ thin skeletal small category

Further examples of categories

Example (Instances)

For $\mathbb{N} = \{0, 1, 2, \ldots\}$ consider:

Further examples of categories

Example (Instances)

For $\mathbb{N} = \{0, 1, 2, \dots\}$ consider:

- $\mathbf{SkeletalFinSets} \coloneqq \dot{\bigcup}_{m,n \in \mathbb{N}} \{1, \dots, n\}^m$

Further examples of categories

Example (Instances)

For $\mathbb{N} = \{0, 1, 2, \ldots\}$ consider:

- $\mathbf{SkeletalFinSets} \coloneqq \dot{\bigcup}_{m,n\in\mathbb{N}}\{1, \ldots, n\}^m$, equivalent, even a computational model for the category of **finite sets**

Further examples of categories

Example (Instances)

For $\mathbb{N} = \{0, 1, 2, \ldots\}$ consider:

- $\mathbf{SkeletalFinSets} := \dot{\bigcup}_{m,n \in \mathbb{N}} \{1, \ldots, n\}^m$, equivalent, even a computational model for the category of **finite sets**
- $\mathbf{\Delta} := \dot{\bigcup}_{m,n \in \mathbb{N}} \underbrace{\{0 < \ldots < n\}^m}_{\text{ordered tuples}}$

Further examples of categories

Example (Instances)

For $\mathbb{N} = \{0, 1, 2, \dots\}$ consider:

- $\mathbf{SkeletalFinSets} \coloneqq \dot{\bigcup}_{m,n\in\mathbb{N}} \{1, \dots, n\}^m$, equivalent, even a computational model for the category of **finite sets**
- $\mathbf{\Delta} \coloneqq \dot{\bigcup}_{m,n\in\mathbb{N}} \underbrace{\{0 < \dots < n\}^m}_{\text{ordered tuples}}$, the **simplicial category**

Further examples of categories

Example (Instances)

For $\mathbb{N} = \{0, 1, 2, \ldots\}$ consider:

- $\mathbf{SkeletalFinSets} \coloneqq \dot{\bigcup}_{m,n \in \mathbb{N}} \{1, \ldots, n\}^m$, equivalent, even a computational model for the category of **finite sets**
- $\mathbf{\Delta} \coloneqq \dot{\bigcup}_{m,n \in \mathbb{N}} \underbrace{\{0 < \ldots < n\}^m}_{\text{ordered tuples}}$, the **simplicial category**

Example (Category constructors)

- **Opposite category** $\mathcal{C}^{\mathrm{op}}$:
  $\mathcal{C}_0^{\mathrm{op}} = \mathcal{C}_0$, $\mathrm{Hom}_{\mathcal{C}^{\mathrm{op}}}(N, M) \coloneqq \mathrm{Hom}_{\mathcal{C}}(M, N)$

Further examples of categories

Example (Instances)

For $\mathbb{N} = \{0, 1, 2, \dots\}$ consider:

- $\mathbf{SkeletalFinSets} := \dot{\bigcup}_{m,n \in \mathbb{N}} \{1, \dots, n\}^m$, equivalent, even a computational model for the category of **finite sets**
- $\boldsymbol{\Delta} := \dot{\bigcup}_{m,n \in \mathbb{N}} \underbrace{\{0 < \dots < n\}^m}_{\text{ordered tuples}}$, the **simplicial category**

Example (Category constructors)

- **Opposite category** $\mathcal{C}^{\mathrm{op}}$:
  $\mathcal{C}_0^{\mathrm{op}} = \mathcal{C}_0$, $\mathrm{Hom}_{\mathcal{C}^{\mathrm{op}}}(N, M) := \mathrm{Hom}_{\mathcal{C}}(M, N)$
- **Free category** $\mathbf{CatClosure}(q)$ generated by a quiver q

Further examples of categories

Example (Instances)

For $\mathbb{N} = \{0, 1, 2, \ldots\}$ consider:

- $\mathbf{SkeletalFinSets} \coloneqq \dot{\bigcup}_{m,n \in \mathbb{N}} \{1, \ldots, n\}^m$, equivalent, even a computational model for the category of **finite sets**
- $\mathbf{\Delta} \coloneqq \dot{\bigcup}_{m,n \in \mathbb{N}} \underbrace{\{0 < \ldots < n\}^m}_{\text{ordered tuples}}$, the **simplicial category**

Example (Category constructors)

- **Opposite category** $\mathcal{C}^{\mathrm{op}}$:
  $\mathcal{C}_0^{\mathrm{op}} = \mathcal{C}_0$, $\mathrm{Hom}_{\mathcal{C}^{\mathrm{op}}}(N, M) \coloneqq \mathrm{Hom}_{\mathcal{C}}(M, N)$
- **Free category** $\mathbf{CatClosure}(q)$ generated by a quiver q
  - $\mathbf{CatClosure}(\bullet) = \circlearrowright \bullet$

Further examples of categories

Example (Instances)

For $\mathbb{N} = \{0, 1, 2, \ldots\}$ consider:

- $\mathbf{SkeletalFinSets} \coloneqq \dot{\bigcup}_{m,n\in\mathbb{N}} \{1, \ldots, n\}^m$, equivalent, even a computational model for the category of **finite sets**
- $\mathbf{\Delta} \coloneqq \dot{\bigcup}_{m,n\in\mathbb{N}} \underbrace{\{0 < \ldots < n\}^m}_{\text{ordered tuples}}$, the **simplicial category**

Example (Category constructors)

- **Opposite category** $\mathcal{C}^{\mathrm{op}}$:
  $\mathcal{C}_0^{\mathrm{op}} = \mathcal{C}_0$, $\mathrm{Hom}_{\mathcal{C}^{\mathrm{op}}}(N, M) \coloneqq \mathrm{Hom}_{\mathcal{C}}(M, N)$
- **Free category** $\mathbf{CatClosure}(q)$ generated by a quiver q
  - $\mathbf{CatClosure}(\bullet) = \circlearrowright \bullet$
  - $\mathbf{CatClosure}(\bullet \to \star) = \circlearrowright \bullet \to \star \circlearrowleft$

Further examples of categories

Example (Instances)

For $\mathbb{N} = \{0, 1, 2, \dots\}$ consider:

- $\mathbf{SkeletalFinSets} \coloneqq \dot{\bigcup}_{m,n \in \mathbb{N}} \{1, \dots, n\}^m$, equivalent, even a computational model for the category of **finite sets**
- $\mathbf{\Delta} \coloneqq \dot{\bigcup}_{m,n \in \mathbb{N}} \underbrace{\{0 < \dots < n\}^m}_{\text{ordered tuples}}$, the **simplicial category**

Example (Category constructors)

- **Opposite category** $\mathcal{C}^{\mathrm{op}}$:
  $\mathcal{C}_0^{\mathrm{op}} = \mathcal{C}_0$, $\mathrm{Hom}_{\mathcal{C}^{\mathrm{op}}}(N, M) \coloneqq \mathrm{Hom}_{\mathcal{C}}(M, N)$
- **Free category** $\mathbf{CatClosure}(q)$ generated by a quiver q
  - $\mathbf{CatClosure}(\bullet) = \circlearrowright \bullet$
  - $\mathbf{CatClosure}(\bullet \to \star) = \circlearrowright \bullet \to \star \circlearrowleft$
  - $\mathbf{CatClosure}(\bullet \to \star \to \blacksquare) = ?$

Further examples of categories

Example (Instances)

For $\mathbb{N} = \{0, 1, 2, \ldots\}$ consider:

- $\mathbf{SkeletalFinSets} := \dot{\bigcup}_{m,n \in \mathbb{N}} \{1, \ldots, n\}^m$, equivalent, even a computational model for the category of **finite sets**
- $\mathbf{\Delta} := \dot{\bigcup}_{m,n \in \mathbb{N}} \underbrace{\{0 < \ldots < n\}^m}_{\text{ordered tuples}}$, the **simplicial category**

Example (Category constructors)

- **Opposite category** $\mathcal{C}^{\mathrm{op}}$:
  $\mathcal{C}_0^{\mathrm{op}} = \mathcal{C}_0$, $\mathrm{Hom}_{\mathcal{C}^{\mathrm{op}}}(N, M) := \mathrm{Hom}_{\mathcal{C}}(M, N)$
- **Free category** $\mathbf{CatClosure}(q)$ generated by a quiver q
  - $\mathbf{CatClosure}(\bullet) = \circlearrowright \bullet$
  - $\mathbf{CatClosure}(\bullet \to \star) = \circlearrowright \bullet \to \star \circlearrowleft$
  - $\mathbf{CatClosure}(\bullet \to \star \to \blacksquare) = ?$
  - $\mathbf{CatClosure}(\bullet \rightleftarrows \star \to \blacksquare) = ?$

Further doctrines: Pre-additive categories

Definition

A category $\mathcal{C}$ is called **pre-additive** if

Definition

A category $\mathcal{C}$ is called **pre-additive** if

- all $\mathrm{Hom}_{\mathcal{C}}(M, N)$ are Abelian groups;

Definition

A category $\mathcal{C}$ is called **pre-additive** if

- all $\mathrm{Hom}_{\mathcal{C}}(M, N)$ are Abelian groups;
- the composition μ is bi-additive.

Further doctrines: Pre-additive categories

Definition

A category $\mathcal{C}$ is called **pre-additive** if

- all $\mathrm{Hom}_{\mathcal{C}}(M, N)$ are Abelian groups;
- the composition μ is bi-additive.

A **ringoid** is a small pre-additive category.

Further doctrines: Pre-additive categories

Definition

A category $\mathcal{C}$ is called **pre-additive** if

- all $\mathrm{Hom}_{\mathcal{C}}(M, N)$ are Abelian groups;
- the composition μ is bi-additive.

A **ringoid** is a small pre-additive category.

Example

(associative unital) ring ≡ ringoid on one object

Further doctrines: k-linear categories

Let k be commutative unital ring.

Definition

A category $\mathcal{C}$ is called **k-linear** if

Further doctrines: k-linear categories

Let k be commutative unital ring.

Definition

A category $\mathcal{C}$ is called **k-linear** if

- $\mathcal{C}$ is pre-additive and all $\mathrm{Hom}_{\mathcal{C}}(M, N)$ are k-modules;

Further doctrines: k-linear categories

Let k be commutative unital ring.

Definition

A category $\mathcal{C}$ is called k**-linear** if

- $\mathcal{C}$ is pre-additive and all $\operatorname{Hom}_{\mathcal{C}}(M, N)$ are k-modules;
- the composition μ is k-bi-linear.

Further doctrines: k-linear categories

Let k be commutative unital ring.

Definition

A category $\mathcal{C}$ is called **k-linear** if

- $\mathcal{C}$ is pre-additive and all $\mathrm{Hom}_{\mathcal{C}}(M, N)$ are k-modules;
- the composition μ is k-bi-linear.

A **k-algebroid** is a small k-linear category.

Further doctrines: k-linear categories

Let k be commutative unital ring.

Definition

A category $\mathcal{C}$ is called **k-linear** if

- $\mathcal{C}$ is pre-additive and all $\mathrm{Hom}_{\mathcal{C}}(M, N)$ are k-modules;
- the composition μ is k-bi-linear.

A **k-algebroid** is a small k-linear category.

Example

k-algebra $\equiv$ k-algebroid on one object

Further doctrines: k-linear categories

Let k be commutative unital ring.

Definition

A category $\mathcal{C}$ is called **k-linear** if

- $\mathcal{C}$ is pre-additive and all $\mathrm{Hom}_{\mathcal{C}}(M, N)$ are k-modules;
- the composition μ is k-bi-linear.

A **k-algebroid** is a small k-linear category.

Example

k-algebra $\equiv$ k-algebroid on one object

We now construct on the computer:

$$\mathbb{Q}\text{-}\mathbf{LinClosure}(\mathbf{CatClosure}(1 \underset{a}{\overset{b}{\rightleftarrows}} 2 \xrightarrow{c} 3))$$

Further doctrines: k-linear categories

Let k be commutative unital ring.

Definition

A category $\mathcal{C}$ is called k**-linear** if

- $\mathcal{C}$ is pre-additive and all $\mathrm{Hom}_{\mathcal{C}}(M, N)$ are k-modules;
- the composition μ is k-bi-linear.

A k**-algebroid** is a small k-linear category.

Example

k-algebra $\equiv$ k-algebroid on one object

We now construct on the computer:

$$\mathbb{Q}\text{-}\mathbf{LinClosure}(\mathbf{CatClosure}(1 \underset{a}{\overset{b}{\rightleftarrows}} 2 \xrightarrow{c} 3))$$

In particular, $\mathbf{CatClosure}$ *invents* the word calculus.

Further doctrines: (co)cartesian categories

- A finite **product** of objects in a category $\mathcal{C}$

$$M_1$$

$$\vdots$$

$$M_\ell$$

.

Further doctrines: (co)cartesian categories

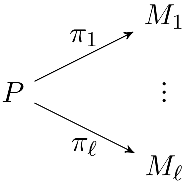
- A finite **product** of objects in a category $\mathcal{C}$

$$
\begin{array}{ccc}
 & & M_1 \\
P & & \vdots \\
 & & M_\ell
\end{array}
$$

.

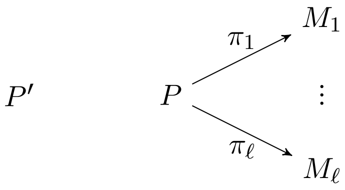
Further doctrines: (co)cartesian categories

- A finite **product** of objects in a category $\mathcal{C}$



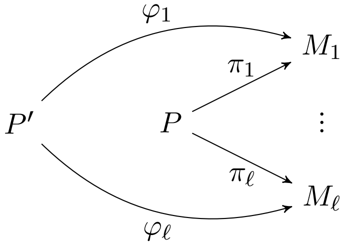
Further doctrines: (co)cartesian categories

- A finite **product** of objects in a category $\mathcal{C}$

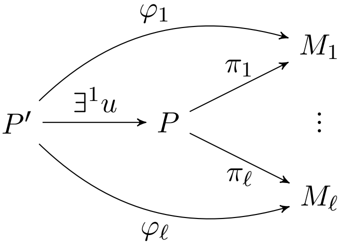


Further doctrines: (co)cartesian categories

- A finite **product** of objects in a category $\mathcal{C}$

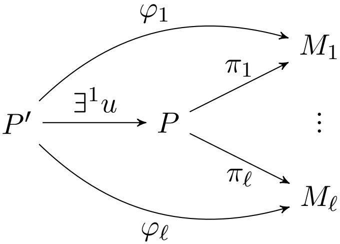


- A finite **product** of objects in a category $\mathcal{C}$



Further doctrines: (co)cartesian categories

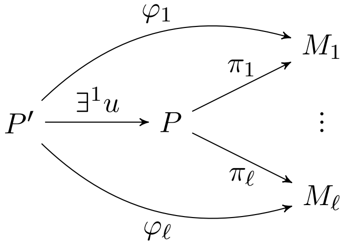
- A finite **product** of objects in a category $\mathcal{C}$



- The empty product is called the **terminal object**;

Further doctrines: (co)cartesian categories

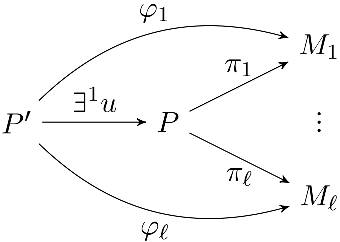
- A finite **product** of objects in a category $\mathcal{C}$



- The empty product is called the **terminal object**;
- A category admitting finite products is called **cartesian**;

Further doctrines: (co)cartesian categories

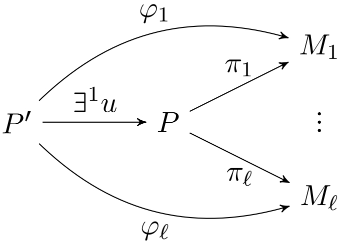
- A finite **product** of objects in a category $\mathcal{C}$



- The empty product is called the **terminal object**;
- A category admitting finite products is called **cartesian**;
- A **coproduct** of objects in $\mathcal{C}$ is a product in $\mathcal{C}^{\mathrm{op}}$;

Further doctrines: (co)cartesian categories

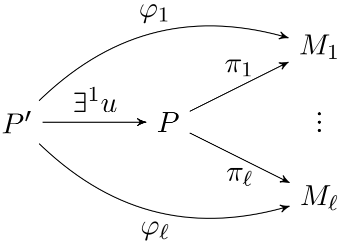
- A finite **product** of objects in a category $\mathcal{C}$



- The empty product is called the **terminal object**;
- A category admitting finite products is called **cartesian**;
- A **coproduct** of objects in $\mathcal{C}$ is a product in $\mathcal{C}^{\mathrm{op}}$;
- The empty coproduct is called the **initial object**;

Further doctrines: (co)cartesian categories

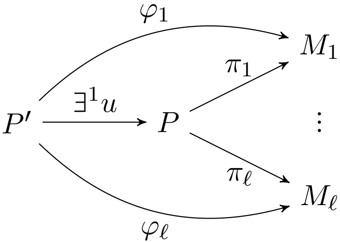
- A finite **product** of objects in a category $\mathcal{C}$



- The empty product is called the **terminal object**;
- A category admitting finite products is called **cartesian**;
- A **coproduct** of objects in $\mathcal{C}$ is a product in $\mathcal{C}^{\mathrm{op}}$;
- The empty coproduct is called the **initial object**;
- A category admitting fin. coproducts is called **cocartesian.**

Further doctrines: (co)cartesian categories

- A finite **product** of objects in a category $\mathcal{C}$



- The empty product is called the **terminal object**;
- A category admitting finite products is called **cartesian**;
- A **coproduct** of objects in $\mathcal{C}$ is a product in $\mathcal{C}^{\mathrm{op}}$;
- The empty coproduct is called the **initial object**;
- A category admitting fin. coproducts is called **cocartesian.**

Q:

What are the initial and terminal objects in $\mathbf{SkeletalFintSets}$?

Additive and k-additive categories

Definition

- A **biproduct** of objects in $\mathcal{C}$ is a product and a coproduct simultaneously

Additive and k-additive categories

Definition

- A **biproduct** of objects in $\mathcal{C}$ is a product and a coproduct simultaneously: We write $M_1 \oplus \cdots \oplus M_\ell$.

Additive and k-additive categories

Definition

- A **biproduct** of objects in $\mathcal{C}$ is a product and a coproduct simultaneously: We write $M_1 \oplus \cdots \oplus M_\ell$.
- A pre-additive category is **additive** if it admits biproducts.

Additive and k-additive categories

Definition

- A **biproduct** of objects in $\mathcal{C}$ is a product and a coproduct simultaneously: We write $M_1 \oplus \cdots \oplus M_\ell$.
- A pre-additive category is **additive** if it admits biproducts.

Example (One more category constructor)

Let R be a k-algebra viewed as a k-linear category:

Additive and k-additive categories

Definition

- A **biproduct** of objects in $\mathcal{C}$ is a product and a coproduct simultaneously: We write $M_1 \oplus \cdots \oplus M_\ell$.
- A pre-additive category is **additive** if it admits biproducts.

Example (One more category constructor)

Let R be a k-algebra viewed as a k-linear category:

$$\mathbf{AdditiveClosure}(R) \coloneqq R\textbf{-mat}$$

.

Additive and k-additive categories

Definition

- A **biproduct** of objects in $\mathcal{C}$ is a product and a coproduct simultaneously: We write $M_1 \oplus \cdots \oplus M_\ell$.
- A pre-additive category is **additive** if it admits biproducts.

Example (One more category constructor)

Let R be a k-algebra viewed as a k-linear category:

$$\mathbf{AdditiveClosure}(R) \coloneqq R\text{-}\mathbf{mat}$$

$$\coloneqq \dot{\bigcup}_{g,g' \in \mathbb{N}} k^{g \times g'}$$

.

Additive and k-additive categories

Definition

- A **biproduct** of objects in $\mathcal{C}$ is a product and a coproduct simultaneously: We write $M_1 \oplus \cdots \oplus M_\ell$.
- A pre-additive category is **additive** if it admits biproducts.

Example (One more category constructor)

Let R be a k-algebra viewed as a k-linear category:

$$\mathbf{AdditiveClosure}(R) \coloneqq R\text{-}\mathbf{mat}$$

$$\coloneqq \dot{\bigcup}_{g,g' \in \mathbb{N}} k^{g \times g'} \coloneqq \begin{cases} \text{Obj:} & g, g', \ldots \in \mathbb{N}, \\ \text{Mor:} & \mathtt{A} \in k^{g \times g'}, \ldots, \end{cases}$$

.

Additive and k-additive categories

Definition

- A **biproduct** of objects in $\mathcal{C}$ is a product and a coproduct simultaneously: We write $M_1 \oplus \cdots \oplus M_\ell$.
- A pre-additive category is **additive** if it admits biproducts.

Example (One more category constructor)

Let R be a k-algebra viewed as a k-linear category:

$$\mathbf{AdditiveClosure}(R) \coloneqq R\text{-}\mathbf{mat}$$

$$\coloneqq \dot{\bigcup}_{g,g' \in \mathbb{N}} k^{g \times g'} \coloneqq \begin{cases} \text{Obj:} & g, g', \ldots \in \mathbb{N}, \\ \text{Mor:} & \mathtt{A} \in k^{g \times g'}, \ldots, \end{cases}$$

with the obvious 4 algorithms for $s, t, 1, \mu$.

Additive and k-additive categories

Definition

- A **biproduct** of objects in $\mathcal{C}$ is a product and a coproduct simultaneously: We write $M_1 \oplus \cdots \oplus M_\ell$.
- A pre-additive category is **additive** if it admits biproducts.

Example (One more category constructor)

Let R be a k-algebra viewed as a k-linear category:

$$\mathbf{AdditiveClosure}(R) \coloneqq R\text{-}\mathbf{mat}$$

$$\coloneqq \dot{\bigcup}_{g,g' \in \mathbb{N}} k^{g \times g'} \coloneqq \begin{cases} \text{Obj:} & g, g', \ldots \in \mathbb{N}, \\ \text{Mor:} & \mathtt{A} \in k^{g \times g'}, \ldots, \end{cases}$$

with the obvious 4 algorithms for $s, t, 1, \mu$.

In particular, $\mathbf{AdditiveClosure}$ *invents* matrix calculus.

How to model the category of finite dimensional vector spaces up to equivalence?

How to model the category of finite dimensional vector spaces up to equivalence?

Example

Let k be a field.

How to model the category of finite dimensional vector spaces up to equivalence?

Example

Let k be a field. Then

$$k\textbf{-vec} := \begin{cases} \text{Obj:} & \text{finite dim. } k\text{-vector spaces,} \\ \text{Mor:} & k\text{-linear maps} \end{cases}$$

How to model the category of finite dimensional vector spaces up to equivalence?

Example

Let k be a field. Then

$$k\textbf{-vec} := \begin{cases} \text{Obj:} & \text{finite dim. } k\text{-vector spaces,} \\ \text{Mor:} & k\text{-linear maps} \end{cases}$$

$$\simeq k\textbf{-mat} = \dot{\bigcup}_{g,g' \in \mathbb{N}} k^{g \times g'}$$

How to model the category of finite dimensional vector spaces up to equivalence?

Example

Let k be a field. Then

$$k\text{-}\mathbf{vec} := \begin{cases} \text{Obj:} & \text{finite dim. } k\text{-vector spaces,} \\ \text{Mor:} & k\text{-linear maps} \end{cases}$$

$$\simeq k\text{-}\mathbf{mat} = \dot{\bigcup}_{g,g' \in \mathbb{N}} k^{g \times g'}$$

$$= \mathbf{AdditiveClosure}(k\text{-}\mathbf{LinClosure}(\mathbf{CatClosure}(\bullet)))$$

How to model the category of finite dimensional vector spaces up to equivalence?

Example

Let k be a field. Then

$$k\text{-}\mathbf{vec} := \begin{cases} \text{Obj:} & \text{finite dim. } k\text{-vector spaces,} \\ \text{Mor:} & k\text{-linear maps} \end{cases}$$
$$\simeq k\text{-}\mathbf{mat} = \dot{\bigcup}_{g,g' \in \mathbb{N}} k^{g \times g'}$$
$$= \mathbf{AdditiveClosure}(k\text{-}\mathbf{LinClosure}(\mathbf{CatClosure}(\bullet)))$$

$k\text{-}\mathbf{vec} \simeq k\text{-}\mathbf{mat}$ has much more structure.

ABELian categories as a specification

An ABELian category is a category in which we can do a very general form of linear algebra.

ABELian categories as a specification

An ABELian category is a category in which we can do a very general form of linear algebra.

Definition

A category $\mathcal{A}$ is called ABELian if

ABELian categories as a specification

An ABELian category is a category in which we can do a very general form of linear algebra.

Definition

A category $\mathcal{A}$ is called ABELian if

- finite biproducts exist,

ABELian categories as a specification

An ABELian category is a category in which we can do a very general form of linear algebra.

Definition

A category $\mathcal{A}$ is called ABELian if

- finite biproducts exist,
- each morphism has an additive inverse,

ABELian categories as a specification

An ABELian category is a category in which we can do a very general form of linear algebra.

Definition

A category $\mathcal{A}$ is called ABELian if

- finite biproducts exist,
- each morphism has an additive inverse, (additive)

ABELian categories as a specification

An ABELian category is a category in which we can do a very general form of linear algebra.

Definition

A category $\mathcal{A}$ is called ABELian if

- finite biproducts exist,
- each morphism has an additive inverse, (additive)
- kernels and cokernels exist,

ABELian categories as a specification

An ABELian category is a category in which we can do a very general form of linear algebra.

Definition

A category $\mathcal{A}$ is called ABELian if

- finite biproducts exist,
- each morphism has an additive inverse, (additive)
- kernels and cokernels exist, (pre-ABELian)

ABELian categories as a specification

An ABELian category is a category in which we can do a very general form of linear algebra.

Definition

A category $\mathcal{A}$ is called ABELian if

- finite biproducts exist,
- each morphism has an additive inverse, (additive)
- kernels and cokernels exist, (pre-ABELian)
- the homomorphism theorem is valid, i.e., $\operatorname{coim}\varphi \xrightarrow{\sim} \operatorname{im}\varphi$.

ABELian categories as a specification

An ABELian category is a category in which we can do a very general form of linear algebra.

Definition

A category $\mathcal{A}$ is called ABELian if

- finite biproducts exist,
- each morphism has an additive inverse, (additive)
- kernels and cokernels exist, (pre-ABELian)
- the homomorphism theorem is valid, i.e., $\operatorname{coim}\varphi \xrightarrow{\sim} \operatorname{im}\varphi$.

Definition

A category is called **computable** ABELian if all disjunctions ($\vee$) and all existential quantifiers ($\exists$) in the axioms of an ABELian category are realized by algorithms.

The “hidden” existential quantifiers of “kernels”

Example

Let $\varphi : M \to N$ be a morphism in $\mathcal{A}$.

$$M \xrightarrow{\varphi} N$$

The “hidden” existential quantifiers of “kernels”

Example

Let $\varphi : M \to N$ be a morphism in $\mathcal{A}$.

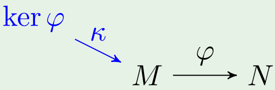
$$\ker \varphi$$

$$M \xrightarrow{\varphi} N$$

The “hidden” existential quantifiers of “kernels”

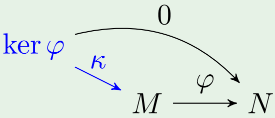
Example

Let $\varphi : M \to N$ be a morphism in $\mathcal{A}$.



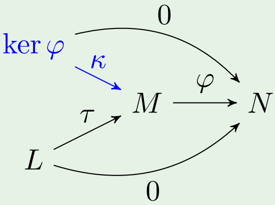
Example

Let $\varphi : M \to N$ be a morphism in $\mathcal{A}$.



Example

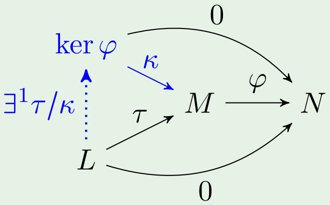
Let $\varphi : M \to N$ be a morphism in $\mathcal{A}$.



The “hidden” existential quantifiers of “kernels”

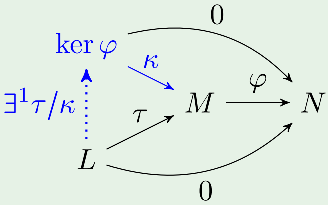
Example

Let $\varphi : M \to N$ be a morphism in $\mathcal{A}$.



Example

Let $\varphi : M \to N$ be a morphism in $\mathcal{A}$.



So $\mathcal{A}$ is a computational context with *many* basic algorithms.

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \mathtt{Source}(\varphi)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \mathtt{Source}(\varphi) \coloneqq \mathtt{NrRows}(\varphi)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \texttt{Source}(\varphi) \coloneqq \texttt{NrRows}(\varphi)$
- $t(\varphi) = \texttt{Range}(\varphi)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \texttt{Source}(\varphi) \coloneqq \texttt{NrRows}(\varphi)$
- $t(\varphi) = \texttt{Range}(\varphi) \coloneqq \texttt{NrColumns}(\varphi)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \texttt{Source}(\varphi) := \texttt{NrRows}(\varphi)$
- $t(\varphi) = \texttt{Range}(\varphi) := \texttt{NrColumns}(\varphi)$
- $1_m = \texttt{IdentityMorphism}(m)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \texttt{Source}(\varphi) \coloneqq \texttt{NrRows}(\varphi)$
- $t(\varphi) = \texttt{Range}(\varphi) \coloneqq \texttt{NrColumns}(\varphi)$
- $1_m = \texttt{IdentityMorphism}(m) \coloneqq \texttt{IdentityMat}(m, k)$

Categorical algorithms of $k\text{-}\mathbf{mat}$

Proposition

$k\text{-}\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \mathtt{Source}(\varphi) \coloneqq \mathtt{NrRows}(\varphi)$
- $t(\varphi) = \mathtt{Range}(\varphi) \coloneqq \mathtt{NrColumns}(\varphi)$
- $1_m = \mathtt{IdentityMorphism}(m) \coloneqq \mathtt{IdentityMat}(m, k)$
- $\varphi\psi = \mathtt{PreCompose}(\varphi, \psi)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \texttt{Source}(\varphi) \coloneqq \texttt{NrRows}(\varphi)$
- $t(\varphi) = \texttt{Range}(\varphi) \coloneqq \texttt{NrColumns}(\varphi)$
- $1_m = \texttt{IdentityMorphism}(m) \coloneqq \texttt{IdentityMat}(m, k)$
- $\varphi\psi = \texttt{PreCompose}(\varphi, \psi) \coloneqq \texttt{MatMul}(\varphi, \psi)$

Categorical algorithms of $k\text{-}\mathbf{mat}$

Proposition

$k\text{-}\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \mathtt{Source}(\varphi) \coloneqq \mathtt{NrRows}(\varphi)$
- $t(\varphi) = \mathtt{Range}(\varphi) \coloneqq \mathtt{NrColumns}(\varphi)$
- $1_m = \mathtt{IdentityMorphism}(m) \coloneqq \mathtt{IdentityMat}(m, k)$
- $\varphi\psi = \mathtt{PreCompose}(\varphi, \psi) \coloneqq \mathtt{MatMul}(\varphi, \psi)$
- $\mathtt{DirectSum}(m, n)$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \texttt{Source}(\varphi) := \texttt{NrRows}(\varphi)$
- $t(\varphi) = \texttt{Range}(\varphi) := \texttt{NrColumns}(\varphi)$
- $1_m = \texttt{IdentityMorphism}(m) := \texttt{IdentityMat}(m, k)$
- $\varphi\psi = \texttt{PreCompose}(\varphi, \psi) := \texttt{MatMul}(\varphi, \psi)$
- $\texttt{DirectSum}(m, n) := m + n$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \texttt{Source}(\varphi) \coloneqq \texttt{NrRows}(\varphi)$
- $t(\varphi) = \texttt{Range}(\varphi) \coloneqq \texttt{NrColumns}(\varphi)$
- $1_m = \texttt{IdentityMorphism}(m) \coloneqq \texttt{IdentityMat}(m, k)$
- $\varphi\psi = \texttt{PreCompose}(\varphi, \psi) \coloneqq \texttt{MatMul}(\varphi, \psi)$
- $\texttt{DirectSum}(m, n) \coloneqq m + n$
- $\texttt{UniversalMorphismIntoDirectSum}(\varphi, \psi)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \texttt{Source}(\varphi) \coloneqq \texttt{NrRows}(\varphi)$
- $t(\varphi) = \texttt{Range}(\varphi) \coloneqq \texttt{NrColumns}(\varphi)$
- $1_m = \texttt{IdentityMorphism}(m) \coloneqq \texttt{IdentityMat}(m, k)$
- $\varphi\psi = \texttt{PreCompose}(\varphi, \psi) \coloneqq \texttt{MatMul}(\varphi, \psi)$
- $\texttt{DirectSum}(m, n) \coloneqq m + n$
- $\texttt{UniversalMorphismIntoDirectSum}(\varphi, \psi) \coloneqq \texttt{Augment}(\varphi, \tau)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \mathtt{Source}(\varphi) \coloneqq \mathtt{NrRows}(\varphi)$
- $t(\varphi) = \mathtt{Range}(\varphi) \coloneqq \mathtt{NrColumns}(\varphi)$
- $1_m = \mathtt{IdentityMorphism}(m) \coloneqq \mathtt{IdentityMat}(m, k)$
- $\varphi\psi = \mathtt{PreCompose}(\varphi, \psi) \coloneqq \mathtt{MatMul}(\varphi, \psi)$
- $\mathtt{DirectSum}(m, n) \coloneqq m + n$
- $\mathtt{UniversalMorphismIntoDirectSum}(\varphi, \psi) \coloneqq \mathtt{Augment}(\varphi, \tau)$
- $\mathtt{UniversalMorphismFromDirectSum}(\varphi, \psi)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \texttt{Source}(\varphi) \coloneqq \texttt{NrRows}(\varphi)$
- $t(\varphi) = \texttt{Range}(\varphi) \coloneqq \texttt{NrColumns}(\varphi)$
- $1_m = \texttt{IdentityMorphism}(m) \coloneqq \texttt{IdentityMat}(m, k)$
- $\varphi\psi = \texttt{PreCompose}(\varphi, \psi) \coloneqq \texttt{MatMul}(\varphi, \psi)$
- $\texttt{DirectSum}(m, n) \coloneqq m + n$
- $\texttt{UniversalMorphismIntoDirectSum}(\varphi, \psi) \coloneqq \texttt{Augment}(\varphi, \tau)$
- $\texttt{UniversalMorphismFromDirectSum}(\varphi, \psi) \coloneqq \texttt{Stack}(\varphi, \tau)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof.

- Objects m, n are natural numbers in $\mathbb{N}$
- Morphisms φ, ψ are rectangular matrices over k
- $s(\varphi) = \texttt{Source}(\varphi) := \texttt{NrRows}(\varphi)$
- $t(\varphi) = \texttt{Range}(\varphi) := \texttt{NrColumns}(\varphi)$
- $1_m = \texttt{IdentityMorphism}(m) := \texttt{IdentityMat}(m, k)$
- $\varphi\psi = \texttt{PreCompose}(\varphi, \psi) := \texttt{MatMul}(\varphi, \psi)$
- $\texttt{DirectSum}(m, n) := m + n$
- $\texttt{UniversalMorphismIntoDirectSum}(\varphi, \psi) := \texttt{Augment}(\varphi, \tau)$
- $\texttt{UniversalMorphismFromDirectSum}(\varphi, \psi) := \texttt{Stack}(\varphi, \tau)$
- ...

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\mathtt{KernelObject}(\varphi)$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\texttt{KernelObject}(\varphi) \coloneqq \texttt{NrRows}(\varphi) - \texttt{Rank}(\varphi)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\texttt{KernelObject}(\varphi) \coloneqq \texttt{NrRows}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{KernelEmbedding}(\varphi)$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\texttt{KernelObject}(\varphi) \coloneqq \texttt{NrRows}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{KernelEmbedding}(\varphi) \coloneqq \texttt{REF}(\texttt{LeftNullSpace}(\varphi))$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\texttt{KernelObject}(\varphi) \coloneqq \texttt{NrRows}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{KernelEmbedding}(\varphi) \coloneqq \texttt{REF}(\texttt{LeftNullSpace}(\varphi))$
- $\texttt{KernelLift}(\varphi, \tau)$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\texttt{KernelObject}(\varphi) \coloneqq \texttt{NrRows}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{KernelEmbedding}(\varphi) \coloneqq \texttt{REF}(\texttt{LeftNullSpace}(\varphi))$
- $\texttt{KernelLift}(\varphi, \tau) \coloneqq \texttt{Sol}(\chi \cdot \texttt{REF}(\texttt{LeftNullSpace}(\varphi)) = \tau)$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\mathtt{KernelObject}(\varphi) \coloneqq \mathtt{NrRows}(\varphi) - \mathtt{Rank}(\varphi)$
- $\mathtt{KernelEmbedding}(\varphi) \coloneqq \mathtt{REF}(\mathtt{LeftNullSpace}(\varphi))$
- $\mathtt{KernelLift}(\varphi, \tau) \coloneqq \mathtt{Sol}(\chi \cdot \mathtt{REF}(\mathtt{LeftNullSpace}(\varphi)) = \tau)$
- $\mathtt{CokernelObject}(\varphi)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\mathtt{KernelObject}(\varphi) \coloneqq \mathtt{NrRows}(\varphi) - \mathtt{Rank}(\varphi)$
- $\mathtt{KernelEmbedding}(\varphi) \coloneqq \mathtt{REF}(\mathtt{LeftNullSpace}(\varphi))$
- $\mathtt{KernelLift}(\varphi, \tau) \coloneqq \mathtt{Sol}(\chi \cdot \mathtt{REF}(\mathtt{LeftNullSpace}(\varphi)) = \tau)$
- $\mathtt{CokernelObject}(\varphi) \coloneqq \mathtt{NrColumns}(\varphi) - \mathtt{Rank}(\varphi)$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\texttt{KernelObject}(\varphi) \coloneqq \texttt{NrRows}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{KernelEmbedding}(\varphi) \coloneqq \texttt{REF}(\texttt{LeftNullSpace}(\varphi))$
- $\texttt{KernelLift}(\varphi, \tau) \coloneqq \texttt{Sol}(\chi \cdot \texttt{REF}(\texttt{LeftNullSpace}(\varphi)) = \tau)$
- $\texttt{CokernelObject}(\varphi) \coloneqq \texttt{NrColumns}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{CokernelProjection}(\varphi)$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\texttt{KernelObject}(\varphi) \coloneqq \texttt{NrRows}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{KernelEmbedding}(\varphi) \coloneqq \texttt{REF}(\texttt{LeftNullSpace}(\varphi))$
- $\texttt{KernelLift}(\varphi, \tau) \coloneqq \texttt{Sol}(\chi \cdot \texttt{REF}(\texttt{LeftNullSpace}(\varphi)) = \tau)$
- $\texttt{CokernelObject}(\varphi) \coloneqq \texttt{NrColumns}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{CokernelProjection}(\varphi) \coloneqq \texttt{CEF}(\texttt{RightNullSpace}(\varphi))$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\texttt{KernelObject}(\varphi) \coloneqq \texttt{NrRows}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{KernelEmbedding}(\varphi) \coloneqq \texttt{REF}(\texttt{LeftNullSpace}(\varphi))$
- $\texttt{KernelLift}(\varphi, \tau) \coloneqq \texttt{Sol}(\chi \cdot \texttt{REF}(\texttt{LeftNullSpace}(\varphi)) = \tau)$
- $\texttt{CokernelObject}(\varphi) \coloneqq \texttt{NrColumns}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{CokernelProjection}(\varphi) \coloneqq \texttt{CEF}(\texttt{RightNullSpace}(\varphi))$
- $\texttt{CokernelColift}(\varphi, \tau)$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\texttt{KernelObject}(\varphi) \coloneqq \texttt{NrRows}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{KernelEmbedding}(\varphi) \coloneqq \texttt{REF}(\texttt{LeftNullSpace}(\varphi))$
- $\texttt{KernelLift}(\varphi, \tau) \coloneqq \texttt{Sol}(\chi \cdot \texttt{REF}(\texttt{LeftNullSpace}(\varphi)) = \tau)$
- $\texttt{CokernelObject}(\varphi) \coloneqq \texttt{NrColumns}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{CokernelProjection}(\varphi) \coloneqq \texttt{CEF}(\texttt{RightNullSpace}(\varphi))$
- $\texttt{CokernelColift}(\varphi, \tau) \coloneqq$
  $\texttt{Sol}(\texttt{CEF}(\texttt{RightNullSpace}(\varphi)) \cdot \chi = \tau)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\texttt{KernelObject}(\varphi) := \texttt{NrRows}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{KernelEmbedding}(\varphi) := \texttt{REF}(\texttt{LeftNullSpace}(\varphi))$
- $\texttt{KernelLift}(\varphi, \tau) := \texttt{Sol}(\chi \cdot \texttt{REF}(\texttt{LeftNullSpace}(\varphi)) = \tau)$
- $\texttt{CokernelObject}(\varphi) := \texttt{NrColumns}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{CokernelProjection}(\varphi) := \texttt{CEF}(\texttt{RightNullSpace}(\varphi))$
- $\texttt{CokernelColift}(\varphi, \tau) :=$
  $\texttt{Sol}(\texttt{CEF}(\texttt{RightNullSpace}(\varphi)) \cdot \chi = \tau)$
- $\texttt{LiftAlongMonomorphism}(\kappa, \tau)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\texttt{KernelObject}(\varphi) \coloneqq \texttt{NrRows}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{KernelEmbedding}(\varphi) \coloneqq \texttt{REF}(\texttt{LeftNullSpace}(\varphi))$
- $\texttt{KernelLift}(\varphi, \tau) \coloneqq \texttt{Sol}(\chi \cdot \texttt{REF}(\texttt{LeftNullSpace}(\varphi)) = \tau)$
- $\texttt{CokernelObject}(\varphi) \coloneqq \texttt{NrColumns}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{CokernelProjection}(\varphi) \coloneqq \texttt{CEF}(\texttt{RightNullSpace}(\varphi))$
- $\texttt{CokernelColift}(\varphi, \tau) \coloneqq$ $\texttt{Sol}(\texttt{CEF}(\texttt{RightNullSpace}(\varphi)) \cdot \chi = \tau)$
- $\texttt{LiftAlongMonomorphism}(\kappa, \tau) \coloneqq \texttt{Sol}(\chi \cdot \kappa = \tau)$

Categorical algorithms of k-$\mathbf{mat}$

Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

Proof. (continued)

- $\texttt{KernelObject}(\varphi) \coloneqq \texttt{NrRows}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{KernelEmbedding}(\varphi) \coloneqq \texttt{REF}(\texttt{LeftNullSpace}(\varphi))$
- $\texttt{KernelLift}(\varphi, \tau) \coloneqq \texttt{Sol}(\chi \cdot \texttt{REF}(\texttt{LeftNullSpace}(\varphi)) = \tau)$
- $\texttt{CokernelObject}(\varphi) \coloneqq \texttt{NrColumns}(\varphi) - \texttt{Rank}(\varphi)$
- $\texttt{CokernelProjection}(\varphi) \coloneqq \texttt{CEF}(\texttt{RightNullSpace}(\varphi))$
- $\texttt{CokernelColift}(\varphi, \tau) \coloneqq$ $\texttt{Sol}(\texttt{CEF}(\texttt{RightNullSpace}(\varphi)) \cdot \chi = \tau)$
- $\texttt{LiftAlongMonomorphism}(\kappa, \tau) \coloneqq \texttt{Sol}(\chi \cdot \kappa = \tau)$
- $\texttt{ColiftAlongEpimorphism}(\varepsilon, \tau)$

Categorical algorithms of k-$\mathbf{mat}$

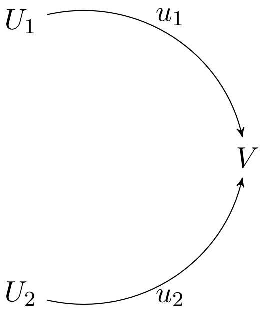
Proposition

k-$\mathbf{mat}$ is a computable Abelian category.

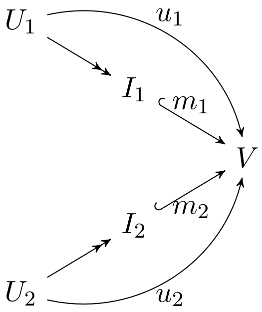
Proof. (continued)

- $\mathtt{KernelObject}(\varphi) \coloneqq \mathtt{NrRows}(\varphi) - \mathtt{Rank}(\varphi)$
- $\mathtt{KernelEmbedding}(\varphi) \coloneqq \mathtt{REF}(\mathtt{LeftNullSpace}(\varphi))$
- $\mathtt{KernelLift}(\varphi, \tau) \coloneqq \mathtt{Sol}(\chi \cdot \mathtt{REF}(\mathtt{LeftNullSpace}(\varphi)) = \tau)$
- $\mathtt{CokernelObject}(\varphi) \coloneqq \mathtt{NrColumns}(\varphi) - \mathtt{Rank}(\varphi)$
- $\mathtt{CokernelProjection}(\varphi) \coloneqq \mathtt{CEF}(\mathtt{RightNullSpace}(\varphi))$
- $\mathtt{CokernelColift}(\varphi, \tau) \coloneqq$ $\mathtt{Sol}(\mathtt{CEF}(\mathtt{RightNullSpace}(\varphi)) \cdot \chi = \tau)$
- $\mathtt{LiftAlongMonomorphism}(\kappa, \tau) \coloneqq \mathtt{Sol}(\chi \cdot \kappa = \tau)$
- $\mathtt{ColiftAlongEpimorphism}(\varepsilon, \tau) \coloneqq \mathtt{Sol}(\varepsilon \cdot \chi = \tau)$ $\square$

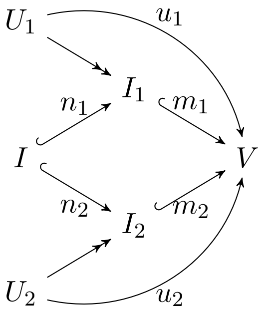
Intersection in Abelian categories



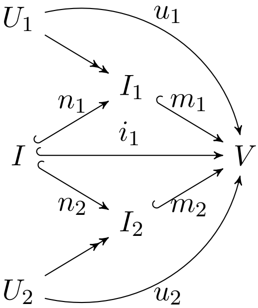
Intersection in Abelian categories



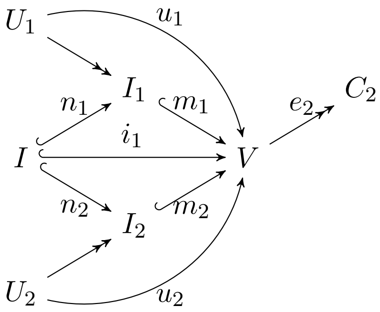
Intersection in Abelian categories



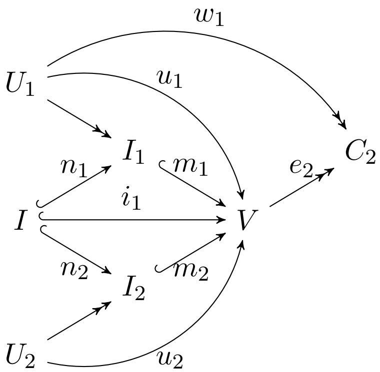
Intersection in Abelian categories



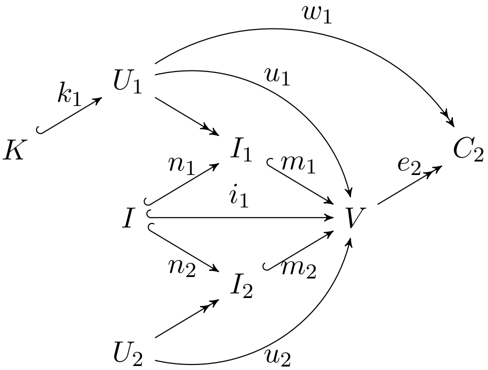
Intersection in Abelian categories



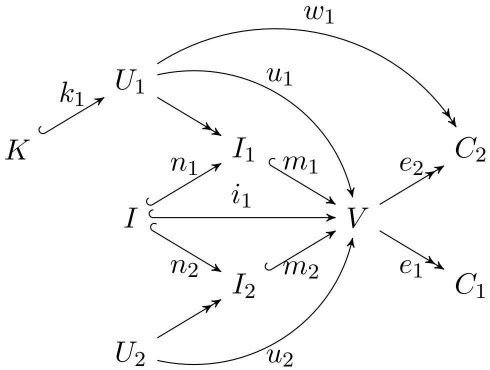
Intersection in Abelian categories



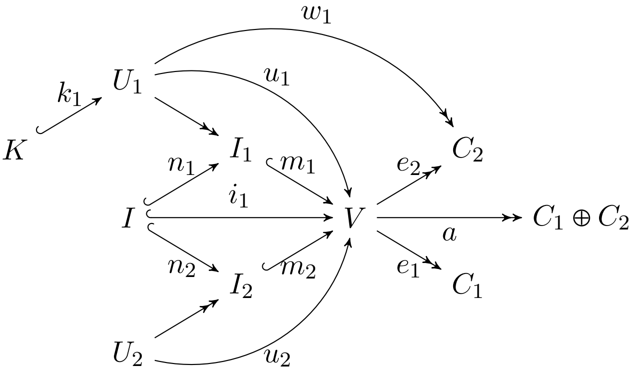
Intersection in Abelian categories



Intersection in Abelian categories



Intersection in Abelian categories



A computable model for R-fdmod

Let k be a field.

A computable model for R-fdmod

Let k be a field.

Proposition (GAP-package `FunctorCategories`)

If $\mathcal{B}$ is a finitely presented k-linear category (k-algebroid) and $\mathcal{A}$ is computable ABELian over k, then the functor category

$$\mathcal{A}^{\mathcal{B}} \coloneqq \mathbf{FuncCat}(\mathcal{B}, \mathcal{A})$$

is computable ABELian over k.

A computable model for R-fdmod

Let k be a field.

Proposition (GAP-package `FunctorCategories`)

If $\mathcal{B}$ is a finitely presented k-linear category (k-algebroid) and $\mathcal{A}$ is computable ABELian over k, then the functor category

$$\mathcal{A}^{\mathcal{B}} \coloneqq \mathbf{FuncCat}(\mathcal{B}, \mathcal{A})$$

is computable ABELian over k.

Corollary

Let R be a finitely presented k-algebra (or k-algebroid), then the category of *finite dimensional* R-modules

$$R\text{-fdmod} \simeq k\text{-}\mathbf{mat}^R = \left(\dot{\bigcup}_{g,g' \in \mathbb{N}} k^{g \times g'}\right)^R$$

is computable ABELian.

A computable model for R-fdmod

Let k be a field.

Proposition (GAP-package FunctorCategories)

If $\mathcal{B}$ is a finitely presented k-linear category (k-algebroid) and $\mathcal{A}$ is computable ABELian over k, then the functor category

$$\mathcal{A}^{\mathcal{B}} := \mathbf{FuncCat}(\mathcal{B}, \mathcal{A})$$

is computable ABELian over k.

Corollary

Let R be a finitely presented k-algebra (or k-algebroid), then the category of *finite dimensional* R-modules

$$R\text{-fdmod} \simeq k\text{-}\mathbf{mat}^{R} = \left(\dot{\bigcup}_{g,g' \in \mathbb{N}} k^{g \times g'}\right)^{R}$$

is computable ABELian.

What about finitely presented modules?

Computable rings

From now on let R be a ring with 1.

Computable rings

From now on let R be a ring with 1.

Definition

We call a constructive ring **left computable** if the solvability of $\mathtt{XA} = \mathtt{B}$ is algorithmically decidable.

Computable rings

From now on let R be a ring with 1.

Definition

We call a constructive ring **left computable** if the solvability of $\mathtt{X}\mathtt{A} = \mathtt{B}$ is algorithmically decidable. This means:

- Determining a **syzygy matrix** $\mathtt{S}$ of $\mathtt{A}$:

$$\mathtt{S}\mathtt{A} = 0, \forall \mathtt{S}' : \mathtt{S}'\mathtt{A} = 0 \implies \exists \mathtt{Y} : \mathtt{Y}\mathtt{S} = \mathtt{S}';$$

Computable rings

From now on let R be a ring with 1.

Definition

We call a constructive ring **left computable** if the solvability of $\mathtt{XA} = \mathtt{B}$ is algorithmically decidable. This means:

- Determining a **syzygy matrix** $\mathtt{S}$ of $\mathtt{A}$:
$$\mathtt{SA} = \mathtt{0}, \forall \mathtt{S}' : \mathtt{S}'\mathtt{A} = \mathtt{0} \implies \exists \mathtt{Y} : \mathtt{YS} = \mathtt{S}';$$
- Deciding the solvability of $\mathtt{XA} = \mathtt{B}$

Computable rings

From now on let R be a ring with 1.

Definition

We call a constructive ring **left computable** if the solvability of $\mathtt{X}\mathtt{A} = \mathtt{B}$ is algorithmically decidable. This means:

- Determining a **syzygy matrix** $\mathtt{S}$ of $\mathtt{A}$:

$$\mathtt{S}\mathtt{A} = 0, \forall \mathtt{S}' : \mathtt{S}'\mathtt{A} = 0 \implies \exists \mathtt{Y} : \mathtt{Y}\mathtt{S} = \mathtt{S}';$$

- Deciding the solvability of $\mathtt{X}\mathtt{A} = \mathtt{B}$ and in the affirmative case determining a **particular solution** $\mathtt{X}$.

Computable rings

From now on let R be a ring with 1.

Definition

We call a constructive ring **left computable** if the solvability of $\mathtt{XA} = \mathtt{B}$ is algorithmically decidable. This means:

- Determining a **syzygy matrix** $\mathtt{S}$ of $\mathtt{A}$:

$$\mathtt{SA} = 0, \forall \mathtt{S}' : \mathtt{S}'\mathtt{A} = 0 \implies \exists \mathtt{Y} : \mathtt{YS} = \mathtt{S}';$$

- Deciding the solvability of $\mathtt{XA} = \mathtt{B}$ and in the affirmative case determining a **particular solution** $\mathtt{X}$.

Proposition ([Pos17])

If R is left computable then the category $\dot{\bigcup}_{g,g' \in \mathbb{N}} R^{g \times g'}$ is computable additive with weak kernels and decidable lifts.

Computable rings

From now on let R be a ring with 1.

Definition

We call a constructive ring **left computable** if the solvability of $\mathtt{XA} = \mathtt{B}$ is algorithmically decidable. This means:

- Determining a **syzygy matrix** $\mathtt{S}$ of $\mathtt{A}$:

$$\mathtt{SA} = 0, \forall \mathtt{S}' : \mathtt{S}'\mathtt{A} = 0 \implies \exists \mathtt{Y} : \mathtt{YS} = \mathtt{S}';$$

- Deciding the solvability of $\mathtt{XA} = \mathtt{B}$ and in the affirmative case determining a **particular solution** $\mathtt{X}$.

Proposition ([Pos17])

If R is left computable then the category $\dot{\bigcup}_{g,g' \in \mathbb{N}} R^{g \times g'}$ is computable additive with weak kernels and decidable lifts.

Now to a computable model for the category of f.p. R-modules:

A computable model for R-fpmod

Freyd construction $\mathbf{Freyd}(\mathbf{P})$

Let $\mathbf{P}$ be an additive category, then a particular ideal quotient

$$\mathbf{Freyd}(\mathbf{P}) := \mathbf{P}^{\{\bullet \to \star\}}/I = \mathbf{FuncCat}(\{\bullet \to \star\}, \mathbf{P})/I$$

is additive *with cokernels*

A computable model for R-$\mathbf{fpmod}$

Freyd construction $\mathbf{Freyd}(\mathbf{P})$

Let $\mathbf{P}$ be an additive category, then a particular ideal quotient

$$\mathbf{Freyd}(\mathbf{P}) := \mathbf{P}^{\{\bullet\to\star\}}/I = \mathbf{FuncCat}(\{\bullet \to \star\}, \mathbf{P})/I$$

is additive *with cokernels*

Theorem ([Pos17])

Freyd's construction yields a computable ABEL*ian category if in addition* $\mathbf{P}$ *has weak cokernels and decidable lifts.*

A computable model for R-$\mathbf{fpmod}$

Freyd construction $\mathbf{Freyd}(\mathbf{P})$

Let $\mathbf{P}$ be an additive category, then a particular ideal quotient

$$\mathbf{Freyd}(\mathbf{P}) := \mathbf{P}^{\{\bullet\to\star\}}/I = \mathbf{FuncCat}(\{\bullet \to \star\}, \mathbf{P})/I$$

is additive *with cokernels*

Theorem ([Pos17])

Freyd's construction yields a computable ABEL*ian category if in addition* $\mathbf{P}$ *has weak cokernels and decidable lifts.*

Corollary ([Pos17], [BLH11])

If R is left computable then

$$R\text{-}\mathbf{fpmod} \simeq \mathbf{Freyd}\left(\dot{\bigcup}_{g,g'\in\mathbb{N}} R^{g\times g'}\right)$$

is computable ABELian.

A computable model for R-$\mathbf{fpmod}$

Freyd construction $\mathbf{Freyd}(\mathbf{P})$

Let $\mathbf{P}$ be an additive category, then a particular ideal quotient

$$\mathbf{Freyd}(\mathbf{P}) := \mathbf{P}^{\{\bullet\to\star\}}/I = \mathbf{FuncCat}(\{\bullet \to \star\}, \mathbf{P})/I$$

is additive *with cokernels*

Theorem ([Pos17])

Freyd's construction yields a computable ABEL*ian category if in addition* $\mathbf{P}$ *has weak cokernels and decidable lifts.*

Corollary ([Pos17], [BLH11])

If R is left computable then

$$R\text{-}\mathbf{fpmod} \simeq \mathbf{Freyd}\left(\dot{\bigcup}_{g,g'\in\mathbb{N}} R^{g\times g'}\right)$$

is computable ABELian.

$\mathbf{Freyd}(\mathbf{AdditiveClosure}(R\text{-}\mathbf{LinClosure}(\mathbf{CatClosure}(\bullet))))$!!

Examples of computable rings

Example (computable rings)

ring	algorithm
a constructive field k	GAUSS
ring of rational integers $\mathbb{Z}$	HERMITE normal form
a univariate polynomial ring $k[x]$	HERMITE normal form
a polynomial ring[a] $R[x_1, \ldots, x_n]$	BUCHBERGER
many noncommutative rings	n.c. BUCHBERGER
$k[x_1, \ldots, x_n]_{\mathfrak{p}}$	~~MORA~~ BUCHBERGER
residue class rings[b]	
...	

[a] R any of the above rings
[b] modulo ideals which are f.g. as left resp. right ideals.

In this context any algorithm to compute a GRÖBNER basis is a substitute for the GAUSS resp. HERMITE normal form algorithm.

Question

Q:

$\mathbf{Freyd}^2(\mathbf{AdditiveClosure}(R\text{-}\mathbf{LinClosure}(\mathbf{CatClosure}(q)..)$?

Question

Q:

$\mathbf{Freyd}^2(\mathbf{AdditiveClosure}(R\text{-}\mathbf{LinClosure}(\mathbf{CatClosure}(q)..)$?

Category theory "invents" data structures and calculi

Free instance of a doctrine	Calculus

Question

Q:

$\mathbf{Freyd}^2(\mathbf{AdditiveClosure}(R\text{-}\mathbf{LinClosure}(\mathbf{CatClosure}(q)..)$?

Category theory "invents" data structures and calculi

Free instance of a doctrine	Calculus
cartesian closed category (CCC)	λ-calculus

Question

Q:

$\mathbf{Freyd}^2(\mathbf{AdditiveClosure}(R\text{-}\mathbf{LinClosure}(\mathbf{CatClosure}(q)..)$?

Category theory “invents” data structures and calculi

Free instance of a doctrine	Calculus
cartesian closed category (CCC)	λ-calculus
compact closed category (CCC)	quantized λ-calculus

Question

Q:

$\mathbf{Freyd}^2(\mathbf{AdditiveClosure}(R\text{-}\mathbf{LinClosure}(\mathbf{CatClosure}(q)..)$?

Category theory "invents" data structures and calculi

Free instance of a doctrine	Calculus
cartesian closed category (CCC)	λ-calculus
compact closed category (CCC)	quantized λ-calculus
topos	non-dependent type theory

Question

Q:

$\mathbf{Freyd}^2(\mathbf{AdditiveClosure}(R\text{-}\mathbf{LinClosure}(\mathbf{CatClosure}(q)..)$?

Category theory "invents" data structures and calculi

Free instance of a doctrine	Calculus
cartesian closed category (CCC)	λ-calculus
compact closed category (CCC)	quantized λ-calculus
topos	non-dependent type theory
locally closed category (LCCC)	dependent type theory

Question

Q:

$\mathbf{Freyd}^2(\mathbf{AdditiveClosure}(R\text{-}\mathbf{LinClosure}(\mathbf{CatClosure}(q)..)$?

Category theory "invents" data structures and calculi

Free instance of a doctrine	Calculus
cartesian closed category (CCC)	λ-calculus
compact closed category (CCC)	quantized λ-calculus
topos	non-dependent type theory
locally closed category (LCCC)	dependent type theory

Software demo

Thank you

- Mohamed Barakat and Markus Lange-Hegermann, *An axiomatic setup for algorithmic homological algebra and an alternative approach to localization*, J. Algebra Appl. **10** (2011), no. 2, 269–293, (`arXiv:1003.1943`). MR 2795737 (2012f:18022)
- Sebastian Posur, *A constructive approach to Freyd categories*, ArXiv e-prints (2017), (`arXiv:1712.03492`).