

# **FinSetsForCAP**

**The elementary topos of (skeletal) finite  
sets**

2020.04.01

1 April 2020

**Mohamed Barakat**

**Julia Mickisch**

**Fabian Zickgraf**

**Mohamed Barakat**

Email: [mohamed.barakat@uni-siegen.de](mailto:mohamed.barakat@uni-siegen.de)

Homepage: <http://www.mathematik.uni-kl.de/~barakat/>

Address: Walter-Flex-Str. 3  
57068 Siegen  
Germany

**Julia Mickisch**

Email: [julia.mickisch@student.uni-siegen.de](mailto:julia.mickisch@student.uni-siegen.de)

Homepage: <https://github.com/juliamick/>

Address: Walter-Flex-Str. 3  
57068 Siegen  
Germany

**Fabian Zickgraf**

Email: [fabian.zickgraf@uni-siegen.de](mailto:fabian.zickgraf@uni-siegen.de)

Homepage: <https://github.com/zickgraf/>

Address: Walter-Flex-Str. 3  
57068 Siegen  
Germany

# Contents

<b>1</b>	<b>The category of finite sets</b>	<b>3</b>
1.1	GAP Categories . . . . .	3
1.2	Attributes . . . . .	3
1.3	Constructors . . . . .	4
1.4	Tools . . . . .	7
1.5	Examples . . . . .	10
<b>2</b>	<b>The category of skeletal finite sets</b>	<b>22</b>
2.1	Skeletal GAP Categories . . . . .	22
2.2	Skeletal Attributes . . . . .	22
2.3	Skeletal Constructors . . . . .	23
2.4	Skeletal Tools . . . . .	24
2.5	Skeletal Examples . . . . .	25
	<b>Index</b>	<b>38</b>

# Chapter 1

## The category of finite sets

### 1.1 GAP Categories

#### 1.1.1 IsFiniteSet (for IsCapCategoryObject)

▷ `IsFiniteSet(object)` (filter)  
**Returns:** true or false  
The GAP category of objects in the category of finite sets.

#### 1.1.2 IsFiniteSetMap (for IsCapCategoryMorphism)

▷ `IsFiniteSetMap(object)` (filter)  
**Returns:** true or false  
The GAP category of morphisms in the category of finite sets.

### 1.2 Attributes

#### 1.2.1 AsList (for IsFiniteSet)

▷ `AsList(M)` (attribute)  
**Returns:** a GAP set  
The GAP set of the list used to construct a finite set  $S$ , i.e., `AsList( FinSet( L ) ) = Set( L )`.

#### 1.2.2 Length (for IsFiniteSet)

▷ `Length(M)` (attribute)  
**Returns:** an integer  
The length of the GAP set of the list used to construct a finite set  $S$ , i.e., `Length( FinSet( L ) ) = Length( Set( L ) )`.

#### 1.2.3 AsList (for IsFiniteSetMap)

▷ `AsList(f)` (attribute)  
**Returns:** a list

The relation underlying a map between finite sets, i.e., `AsList( MapOfFinSets( S, G, T ) ) = G`.

## 1.3 Constructors

### 1.3.1 FinSet (for IsList)

▷ `FinSet(L)` (operation)

**Returns:** a CAP object

Construct a finite set out of the list  $L$ , i.e., an object in the CAP category `FinSets`. The GAP operation `Set` must be applicable to  $L$  without throwing an error. Equality is determined as follows: `FinSet( L1 ) = FinSet( L2 )` iff `IsEqualForElementsOfFinSets( Immutable( Set( L1 ) ), Immutable( Set( L2 ) ) )`. Warning: all internal operations use `FinSetNC` (see below) instead of `FinSet`. Thus, this notion of equality is only valid for objects created by calling `FinSet` explicitly. Internally, `FinSet( L )` is an alias for `FinSetNC( Set( L ) )` and equality is determined as for `FinSetNC`. Thus, `FinSet( L1 ) = FinSetNC( L2 )` iff `IsEqualForElementsOfFinSets( Immutable( Set( L1 ) ), Immutable( L2 ) )` and `FinSetNC( L1 ) = FinSet( L2 )` iff `IsEqualForElementsOfFinSets( Immutable( L1 ), Immutable( Set( L2 ) ) )`.

Example

```
gap> S := FinSet( [ 1, 3, 2, 2, 1 ] );
<An object in FinSets>
gap> Display( S );
[ 1, 2, 3 ]
gap> L := AsList( S );
[ 1, 2, 3 ]
gap> Q := FinSet( L );
<An object in FinSets>
gap> S = Q;
true
gap> FinSet( [ 1, 2 ] ) = FinSet( [ 2, 1 ] );
true
gap> M := FinSetNC( [ , 1, 2, 3 ] );
<An object in FinSets>
gap> IsWellDefined( M );
false
gap> M := FinSetNC( [ 1, 2, 3, 3 ] );
<An object in FinSets>
gap> IsWellDefined( M );
false
```

### 1.3.2 FinSetNC (for IsList)

▷ `FinSetNC(L)` (operation)

**Returns:** a CAP object

Construct a finite set out of the duplicate-free (w.r.t. `IsEqualForElementsOfFinSets`) and dense list  $L$ , i.e., an object in the CAP category `FinSets`. Equality is determined as follows: `FinSetNC( L1 ) = FinSetNC( L2 )` iff `IsEqualForElementsOfFinSets( Immutable( L1 ), Immutable( L2 ) )`.

Example

```

gap> S := FinSetNC( [ 1, 3, 2 ] );
<An object in FinSets>
gap> Display( S );
[ 1, 3, 2 ]
gap> L := AsList( S );
[ 1, 3, 2 ]
gap> Q := FinSetNC( L );
<An object in FinSets>
gap> S = Q;
true
gap> FinSetNC( [ 1, 2 ] ) = FinSetNC( [ 2, 1 ] );
false

```

### 1.3.3 MapOfFinSets (for IsFiniteSet, IsList, IsFiniteSet)

▷ MapOfFinSets( $S$ ,  $G$ ,  $T$ )

(operation)

**Returns:** a CAP morphism

Construct a map  $\phi : S \rightarrow T$  of the finite sets  $S$  and  $T$ , i.e., a morphism in the CAP category FinSets, where  $G$  is a list of pairs in  $S \times T$  describing the graph of  $\phi$ .

Example

```

gap> S := FinSet( [ 1, 3, 2, 2, 1 ] );
<An object in FinSets>
gap> T := FinSet( [ "a", "b", "c" ] );
<An object in FinSets>
gap> G := [ [ 1, "b" ], [ 3, "b" ], [ 2, "a" ] ];
gap> phi := MapOfFinSets( S, G, T );
<A morphism in FinSets>
gap> IsWellDefined( phi );
true
gap> phi( 1 );
"b"
gap> phi( 2 );
"a"
gap> phi( 3 );
"b"
gap> List( S, phi );
[ "b", "a", "b" ]
gap> psi := [ [ 1, "b" ], [ 2, "a" ], [ 3, "b" ] ];
gap> psi := MapOfFinSets( S, psi, T );
<A morphism in FinSets>
gap> IsWellDefined( psi );
true
gap> phi = psi;
true
gap> psi := MapOfFinSetsNC( S, [ , [ 1, "b" ], [ 3, "b" ], [ 2, "a" ] ], T );
<A morphism in FinSets>
gap> IsWellDefined( psi );
false
gap> psi := MapOfFinSets( S, [ [ 1, "d" ], [ 3, "b" ] ], T );
<A morphism in FinSets>

```

```

gap> IsWellDefined( psi );
false
gap> psi := MapOfFinSets( S, [ 1, 2, 3 ], T );
<A morphism in FinSets>
gap> IsWellDefined( psi );
false
gap> psi := MapOfFinSets( S, [ [ 1, "b" ], [ 3, "b" ], [ 2, "a", "b" ] ], T );
<A morphism in FinSets>
gap> IsWellDefined( psi );
false
gap> psi := MapOfFinSets( S, [ [ 5, "b" ], [ 3, "b" ], [ 2, "a" ] ], T );
<A morphism in FinSets>
gap> IsWellDefined( psi );
false
gap> psi := MapOfFinSets( S, [ [ 1, "d" ], [ 3, "b" ], [ 2, "a" ] ], T );
<A morphism in FinSets>
gap> IsWellDefined( psi );
false
gap> psi := MapOfFinSets( S, [ [ 1, "b" ], [ 2, "b" ], [ 2, "a" ] ], T );
<A morphism in FinSets>
gap> IsWellDefined( psi );
false

```

### 1.3.4 MapOfFinSetsNC (for IsFiniteSet, IsList, IsFiniteSet)

▷ MapOfFinSetsNC( $S$ ,  $G$ ,  $T$ )

(operation)

**Returns:** a CAP morphism

Construct a map  $\phi : S \rightarrow T$  of the finite sets  $S$  and  $T$ , i.e., a morphism in the CAP category FinSets, where  $G$  is a duplicate-free and dense list of pairs in  $S \times T$  describing the graph of  $\phi$ .

Example

```

gap> S := FinSetNC( [ 1, 3, 2 ] );
<An object in FinSets>
gap> T := FinSetNC( [ "a", "b", "c" ] );
<An object in FinSets>
gap> G := [ [ 1, "b" ], [ 3, "b" ], [ 2, "a" ] ];
gap> phi := MapOfFinSetsNC( S, G, T );
<A morphism in FinSets>
gap> IsWellDefined( phi );
true
gap> phi( 1 );
"b"
gap> phi( 2 );
"a"
gap> phi( 3 );
"b"
gap> List( S, phi );
[ "b", "b", "a" ]
gap> psi := [ [ 1, "b" ], [ 2, "a" ], [ 3, "b" ] ];
gap> psi := MapOfFinSetsNC( S, psi, T );
<A morphism in FinSets>
gap> IsWellDefined( psi );

```

```

true
gap> phi = psi;
true

```

## 1.4 Tools

### 1.4.1 IsEqualForElementsOfFinSets (for IsObject, IsObject)

▷ IsEqualForElementsOfFinSets(a, b) (operation)

**Returns:** a boolean

Compares two arbitrary objects using the following rules:

- integers, strings and chars are compared using the operation =
- lists and records are compared recursively
- CAP category objects are compared using IsEqualForObjects (if available)
- CAP category morphisms are compared using IsEqualForMorphismsOnMor (if available)
- other objects are compared using IsIdenticalObj

Note: if CAP category objects or CAP category morphisms are compared using IsEqualForObjects or IsEqualForMorphismsOnMor, respectively, the result must not be fail.

Example

```

gap> IsEqualForElementsOfFinSets( 2, 2 );
true
gap> IsEqualForElementsOfFinSets( 2, "2" );
false
gap> IsEqualForElementsOfFinSets( [ 2 ], [ 2 ] );
true
gap> IsEqualForElementsOfFinSets( [ 2 ], [ 2, 3 ] );
false
gap> IsEqualForElementsOfFinSets( [ , 2 ], [ 2, 2 ] );
false
gap> IsEqualForElementsOfFinSets( rec( a := "a", b := "b" ),
>                                rec( b := "b", a := "a" )
>                                );
true
gap> IsEqualForElementsOfFinSets( rec( a := "a", b := "b" ),
>                                rec( a := "a" )
>                                );
false
gap> IsEqualForElementsOfFinSets( rec( a := "a", b := "b" ),
>                                rec( a := "a", b := "notb" )
>                                );
false
gap> M := FinSet( [ ] );;
gap> N := FinSet( [ ] );;
gap> m := FinSet( 0 );;
gap> id_M := IdentityMorphism( M );;
gap> id_N := IdentityMorphism( N );;

```



```

gap> id_m := IdentityMorphism( m );;
gap> IsEqualForElementsOfFinSets( M, N );
true
gap> IsEqualForElementsOfFinSets( M, m );
false
gap> IsEqualForElementsOfFinSets( id_M, id_N );
true
gap> IsEqualForElementsOfFinSets( id_M, id_m );
false
gap> IsEqualForElementsOfFinSets( FinSets, SkeletalFinSets );
false

```

### 1.4.2 `\in` (for `IsObject`, `IsFiniteSet`)

▷ `\in(obj, M)` (operation)

**Returns:** a boolean

Returns true if there exists an element in `AsList( M )` which is equal to `obj` w.r.t. `IsEqualForElementsOfFinSets` and false if not.

### 1.4.3 `[]` (for `IsFiniteSet`, `IsInt`)

▷ `[](M, i)` (operation)

**Returns:** an object

Returns the  $i$ -th entry of the GAP set of the list used to construct a finite set  $S$ , i.e., `FinSet( L ) [ i ] = Set( L ) [ i ]`.

### 1.4.4 `Iterator` (for `IsFiniteSet`)

▷ `Iterator(M)` (operation)

**Returns:** an iterator

An iterator of the GAP set of the list used to construct a finite set  $S$ , i.e., `Iterator( FinSet( L ) ) = Iterator( Set( L ) )`.

### 1.4.5 `UnionOfFinSets` (for `IsList`)

▷ `UnionOfFinSets(L)` (operation)

**Returns:** a CAP object

Compute the set-theoretic union of the elements of  $L$ , where  $L$  is a list of finite sets.

### 1.4.6 `ListOp` (for `IsFiniteSet`, `IsFunction`)

▷ `ListOp(M, f)` (operation)

**Returns:** a list

Returns `List( AsList( M ), f )`.

### 1.4.7 `FilteredOp` (for `IsFiniteSet`, `IsFunction`)

▷ `FilteredOp(M, f)` (operation)

**Returns:** a list

Returns `FinSetNC( Filtered( AsList( M ), f ) )`.

#### 1.4.8 FirstOp (for IsFiniteSet, IsFunction)

- ▷ `FirstOp(M, f)` (operation)  
**Returns:** a list  
 Returns `First( AsList( M ), f )`.

#### 1.4.9 EmbeddingOfFinSets (for IsFiniteSet, IsFiniteSet)

- ▷ `EmbeddingOfFinSets(S, T)` (operation)  
**Returns:** a CAP morphism  
 Construct the embedding  $\iota : S \rightarrow T$  of the finite sets  $S$  and  $T$ , where  $S$  must be subset of  $T$ .

#### 1.4.10 ProjectionOfFinSets (for IsFiniteSet, IsFiniteSet)

- ▷ `ProjectionOfFinSets(S, T)` (operation)  
**Returns:** a CAP morphism  
 Construct the projection  $\pi : S \rightarrow T$  of the finite sets  $S$  and  $T$ , where  $T$  is a partition of  $S$ .

#### 1.4.11 Preimage (for IsFiniteSetMap, IsFiniteSet)

- ▷ `Preimage(f, T_)` (operation)  
**Returns:** a CAP object  
 Compute the preimage of  $T_$  under the morphism  $f$ .

#### 1.4.12 ImageObject (for IsFiniteSetMap, IsFiniteSet)

- ▷ `ImageObject(f, S_)` (operation)  
**Returns:** a CAP object  
 Compute the image of  $S_$  under the morphism  $f$ .

#### 1.4.13 CallFuncList (for IsFiniteSetMap, IsList)

- ▷ `CallFuncList(phi, L)` (operation)  
**Returns:** a list  
 Returns the image of  $L[1]$  under the map  $\phi$  assuming  $L[1]$  is an element of `AsList( Source( phi ) )`.

#### 1.4.14 ListOp (for IsFiniteSet, IsFiniteSetMap)

- ▷ `ListOp(F, phi)` (operation)  
**Returns:** a list  
 Returns `List( AsList( F ), phi )`.

## 1.5 Examples

### 1.5.1 IsHomSetInhabited

Example

```
gap> L := FinSet( [ ] );
<An object in FinSets>
gap> M := FinSet( [ 2 ] );
<An object in FinSets>
gap> N := FinSet( [ 3 ] );
<An object in FinSets>
gap> IsHomSetInhabited( L, L );
true
gap> IsHomSetInhabited( M, L );
false
gap> IsHomSetInhabited( L, M );
true
gap> IsHomSetInhabited( M, N );
true
```

### 1.5.2 PreCompose

Example

```
gap> S := FinSet( [ 1, 2, 3 ] );
<An object in FinSets>
gap> T := FinSet( [ "a", "b" ] );
<An object in FinSets>
gap> phi := [ [ 1, "b" ], [ 2, "a" ], [ 3, "b" ] ];
gap> phi := MapOfFinSets( S, phi, T );
<A morphism in FinSets>
gap> psi := [ [ "a", 3 ], [ "b", 1 ] ];
gap> psi := MapOfFinSets( T, psi, S );
<A morphism in FinSets>
gap> alpha := PreCompose( phi, psi );
<A morphism in FinSets>
gap> List( S, alpha );
[ 1, 3, 1 ]
gap> IsOne( alpha );
false
```

### 1.5.3 IsEpimorphism and IsMonomorphism

Example

```
gap> S := FinSet( [ 1, 2, 3 ] );
<An object in FinSets>
gap> T := S;
<An object in FinSets>
gap> phi := [ [ 1, 2 ], [ 2, 3 ], [ 3, 1 ] ];
gap> phi := MapOfFinSets( S, phi, T );
<A morphism in FinSets>
gap> I := ImageObject( phi );
<An object in FinSets>
gap> Length( I );
3
```

```

gap> IsMonomorphism( phi );
true
gap> IsSplitMonomorphism( phi );
true
gap> IsEpimorphism( phi );
true
gap> IsSplitEpimorphism( phi );
true
gap> iota := ImageEmbedding( phi );
<A monomorphism in FinSets>
gap> pi := CostrictionToImage( phi );
<An epimorphism in FinSets>
gap> PreCompose( pi, iota ) = phi;
true

```

### 1.5.4 Initial and Terminal Objects

Example

```

gap> M := FinSet( [ 1, 2, 3 ] );
<An object in FinSets>
gap> IsInitial( M );
false
gap> IsTerminal( M );
false
gap> I := InitialObject( M );
<An object in FinSets>
gap> IsInitial( I );
true
gap> IsTerminal( I );
false
gap> iota := UniversalMorphismFromInitialObject( M );
<A morphism in FinSets>
gap> Display( I );
[ ]
gap> T := TerminalObject( M );
<An object in FinSets>
gap> Display( T );
[ "*" ]
gap> IsInitial( T );
false
gap> IsTerminal( T );
true
gap> pi := UniversalMorphismIntoTerminalObject( M );
<A morphism in FinSets>
gap> IsIdenticalObj( Range( pi ), T );
true
gap> t := FinSet( [ "Julia" ] );
<An object in FinSets>
gap> pi_t := UniversalMorphismIntoTerminalObjectWithGivenTerminalObject( M, t );
<A morphism in FinSets>
gap> List( M, pi_t );
[ "Julia", "Julia", "Julia" ]

```

### 1.5.5 Projective and Injective Objects

Example

```
gap> I := FinSet( [ ] );
<An object in FinSets>
gap> T := FinSet( [ 1 ] );
<An object in FinSets>
gap> M := FinSet( [ 2 ] );
<An object in FinSets>
gap> IsProjective( I );
true
gap> IsProjective( T );
true
gap> IsProjective( M );
true
gap> IsOne( EpimorphismFromSomeProjectiveObject( I ) );
true
gap> IsOne( EpimorphismFromSomeProjectiveObject( M ) );
true
```

Example

```
gap> I := FinSet( [ ] );
<An object in FinSets>
gap> T := FinSet( [ 1 ] );
<An object in FinSets>
gap> M := FinSet( [ 2 ] );
<An object in FinSets>
gap> IsInjective( I );
false
gap> IsInjective( T );
true
gap> IsInjective( M );
true
gap> IsIsomorphism( MonomorphismIntoSomeInjectiveObject( I ) );
false
gap> IsMonomorphism( MonomorphismIntoSomeInjectiveObject( I ) );
true
gap> IsOne( MonomorphismIntoSomeInjectiveObject( M ) );
true
```

### 1.5.6 Product

Example

```
gap> S := FinSet( [ 1, 2, 3 ] );
<An object in FinSets>
gap> Length( S );
3
gap> T := FinSet( [ "a", "b" ] );
<An object in FinSets>
gap> Length( T );
2
gap> P := DirectProduct( S, T );
<An object in FinSets>
gap> Length( P );
6
```

```
gap> Display( P );
[ [ 1, "a" ], [ 1, "b" ], [ 2, "a" ], [ 2, "b" ], [ 3, "a" ], [ 3, "b" ] ]
```

### 1.5.7 Coproduct

Example

```
gap> S := FinSet( [ 1, 2, 3 ] );
<An object in FinSets>
gap> Length( S );
3
gap> T := FinSet( [ "a", "b" ] );
<An object in FinSets>
gap> Length( T );
2
gap> C := Coproduct( T, S );
<An object in FinSets>
gap> Length( C );
5
gap> Display( C );
[ [ 1, "a" ], [ 1, "b" ], [ 2, 1 ], [ 2, 2 ], [ 2, 3 ] ]
gap> M := FinSet( [ 1, 2, 3, 4, 5, 6, 7 ] );
<An object in FinSets>
gap> N := FinSet( [ 1, 2, 3 ] );
<An object in FinSets>
gap> P := FinSet( [ 1, 2, 3, 4 ] );
<An object in FinSets>
gap> C := Coproduct( M, N, P );
<An object in FinSets>
gap> AsList( C );
[ [ 1, 1 ], [ 1, 2 ], [ 1, 3 ], [ 1, 4 ], [ 1, 5 ], [ 1, 6 ],
  [ 1, 7 ], [ 2, 1 ], [ 2, 2 ], [ 2, 3 ], [ 3, 1 ], [ 3, 2 ],
  [ 3, 3 ], [ 3, 4 ] ]
gap> iota1 := InjectionOfCofactorOfCoproduct( [ M, N, P ], 1 );
<A morphism in FinSets>
gap> IsWellDefined( iota1 );
true
gap> AsList( iota1 );
[ [ 1, [ 1, 1 ] ], [ 2, [ 1, 2 ] ], [ 3, [ 1, 3 ] ], [ 4, [ 1, 4 ] ],
  [ 5, [ 1, 5 ] ], [ 6, [ 1, 6 ] ], [ 7, [ 1, 7 ] ] ]
gap> iota2 := InjectionOfCofactorOfCoproduct( [ M, N, P ], 2 );
<A morphism in FinSets>
gap> IsWellDefined( iota2 );
true
gap> AsList( iota2 );
[ [ 1, [ 2, 1 ] ], [ 2, [ 2, 2 ] ], [ 3, [ 2, 3 ] ] ]
gap> iota3 := InjectionOfCofactorOfCoproduct( [ M, N, P ], 3 );
<A morphism in FinSets>
gap> IsWellDefined( iota3 );
true
gap> AsList( iota3 );
[ [ 1, [ 3, 1 ] ], [ 2, [ 3, 2 ] ], [ 3, [ 3, 3 ] ], [ 4, [ 3, 4 ] ] ]
gap> psi := UniversalMorphismFromCoproduct( [ M, N, P ],
> [ iota1, iota2, iota3 ]
```

```

>
);
<A morphism in FinSets>
gap> psi = IdentityMorphism( Coproduct( [ M, N, P ] ) );
true

```

### 1.5.8 Image

Example

```

gap> S := FinSet( [ 1, 2, 3 ] );
<An object in FinSets>
gap> T := FinSet( [ "a", "b", "c" ] );
<An object in FinSets>
gap> phi := [ [ 1, "b" ], [ 2, "a" ], [ 3, "b" ] ];;
gap> phi := MapOfFinSets( S, phi, T );
<A morphism in FinSets>
gap> I := ImageObject( phi );
<An object in FinSets>
gap> Length( I );
2
gap> IsMonomorphism( phi );
false
gap> IsSplitMonomorphism( phi );
false
gap> IsEpimorphism( phi );
false
gap> IsSplitEpimorphism( phi );
false
gap> iota := ImageEmbedding( phi );
<A monomorphism in FinSets>
gap> pi := CostrictionToImage( phi );
<An epimorphism in FinSets>
gap> PreCompose( pi, iota ) = phi;
true

```

### 1.5.9 Coimage

Example

```

gap> S := FinSet( [ 1, 2, 3 ] );
<An object in FinSets>
gap> T := FinSet( [ "a", "b", "c" ] );
<An object in FinSets>
gap> phi := [ [ 1, "b" ], [ 2, "a" ], [ 3, "b" ] ];;
gap> phi := MapOfFinSets( S, phi, T );
<A morphism in FinSets>
gap> I := Coimage( phi );
<An object in FinSets>
gap> Length( I );
2
gap> IsMonomorphism( phi );
false
gap> IsSplitMonomorphism( phi );
false
gap> IsEpimorphism( phi );

```

```

false
gap> IsSplitEpimorphism( phi );
false
gap> iota := AstrictionToCoimage( phi );
<A monomorphism in FinSets>
gap> pi := CoimageProjection( phi );
<An epimorphism in FinSets>
gap> PreCompose( pi, iota ) = phi;
true

```

### 1.5.10 Equalizer

Example

```

gap> S := FinSet( [ 1 .. 5 ] );
<An object in FinSets>
gap> T := FinSet( [ 1 .. 3 ] );
<An object in FinSets>
gap> f1 := MapOfFinSets( S, [ [1,3],[2,3],[3,1],[4,2],[5,2] ], T );
<A morphism in FinSets>
gap> f2 := MapOfFinSets( S, [ [1,3],[2,2],[3,3],[4,1],[5,2] ], T );
<A morphism in FinSets>
gap> f3 := MapOfFinSets( S, [ [1,3],[2,1],[3,2],[4,1],[5,2] ], T );
<A morphism in FinSets>
gap> D := [ f1, f2, f3 ];
[ <A morphism in FinSets>, <A morphism in FinSets>, <A morphism in FinSets> ]
gap> Eq := Equalizer( D );
<An object in FinSets>
gap> Display( Eq );
[ 1, 5 ]
gap> iota := EmbeddingOfEqualizer( D );
gap> IsWellDefined( iota );
true
gap> Im := ImageObject( iota );
<An object in FinSets>
gap> Display( Im );
[ 1, 5 ]
gap> mu := MorphismFromEqualizerToSink( D );
gap> PreCompose( iota, f1 ) = mu;
true
gap> M := FinSet( [ "a" ] );
gap> phi := MapOfFinSets( M, [ [ "a", 5 ] ], S );
gap> IsWellDefined( phi );
true
gap> psi := UniversalMorphismIntoEqualizer( D, phi );
<A morphism in FinSets>
gap> IsWellDefined( psi );
true
gap> Display( psi );
[ [ "a" ], [ [ "a", 5 ] ], [ 1, 5 ] ]
gap> PreCompose( psi, iota ) = phi;
true

```



### 1.5.11 Pullback

Example

```
gap> M := FinSet( [ 1 .. 5 ] );
<An object in FinSets>
gap> N1 := FinSet( [ 1 .. 3 ] );
<An object in FinSets>
gap> iota1 := EmbeddingOfFinSets( N1, M );
<A monomorphism in FinSets>
gap> N2 := FinSet( [ 2 .. 5 ] );
<An object in FinSets>
gap> iota2 := EmbeddingOfFinSets( N2, M );
<A monomorphism in FinSets>
gap> D := [ iota1, iota2 ];
[ <A monomorphism in FinSets>, <A monomorphism in FinSets> ]
gap> int := FiberProduct( D );
<An object in FinSets>
gap> Display( int );
[ [ 2, 2 ], [ 3, 3 ] ]
gap> pi1 := ProjectionInFactorOfFiberProduct( D, 1 );
<A monomorphism in FinSets>
gap> int1 := ImageObject( pi1 );
<An object in FinSets>
gap> Display( int1 );
[ 2, 3 ]
gap> pi2 := ProjectionInFactorOfFiberProduct( D, 2 );
<A monomorphism in FinSets>
gap> int2 := ImageObject( pi2 );
<An object in FinSets>
gap> Display( int2 );
[ 2, 3 ]
```

### 1.5.12 Coequalizer

Example

```
gap> N := FinSet( [1,3] );
<An object in FinSets>
gap> M := FinSet( [1,2,4] );
<An object in FinSets>
gap> f := MapOfFinSets( N, [ [1,1], [3,2] ], M );
<A morphism in FinSets>
gap> g := MapOfFinSets( N, [ [1,2], [3,4] ], M );
<A morphism in FinSets>
gap> C := Coequalizer( f, g );
<An object in FinSets>
gap> AsList( C );
[ [ 1, 2, 4 ] ]
gap> A := FinSet( [ 1, 2, 3, 4 ] );
<An object in FinSets>
gap> B := FinSet( [ 1, 2, 3, 4, 5, 6, 7, 8 ] );
<An object in FinSets>
gap> f1 := MapOfFinSets( A, [ [ 1, 1 ], [ 2, 2 ], [ 3, 3 ], [ 4, 8 ] ], B );
<A morphism in FinSets>
gap> f2 := MapOfFinSets( A, [ [ 1, 2 ], [ 2, 3 ], [ 3, 8 ], [ 4, 5 ] ], B );
```

```

<A morphism in FinSets>
gap> f3 := MapOfFinSets( A, [ [ 1, 4 ], [ 2, 2 ], [ 3, 3 ], [ 4, 8 ] ], B );
<A morphism in FinSets>
gap> C1 := Coequalizer( [ f1, f3 ] );
<An object in FinSets>
gap> AsList( C1 );
[ [ 1, 4 ], [ 2 ], [ 3 ], [ 5 ], [ 6 ], [ 7 ], [ 8 ] ]
gap> C2 := Coequalizer( [ f1, f2, f3 ] );
<An object in FinSets>
gap> AsList( C2 );
[ [ 1, 2, 3, 8, 5, 4 ], [ 6 ], [ 7 ] ]
gap> S := FinSet( [ 1 .. 5 ] );
<An object in FinSets>
gap> T := FinSet( [ 1 .. 4 ] );
<An object in FinSets>
gap> f := MapOfFinSets( S, [ [1,2], [2,4], [3,4], [4,3], [5,4] ], T );
<A morphism in FinSets>
gap> g := MapOfFinSets( S, [ [1,2], [2,3], [3,4], [4,3], [5,4] ], T );
<A morphism in FinSets>
gap> C := Coequalizer( f, g );
<An object in FinSets>
gap> Display( C );
[ [ 1 ], [ 2 ], [ 4, 3 ] ]
gap> S := FinSet( [ 1, 2, 3, 4, 5 ] );
<An object in FinSets>
gap> T := FinSet( [ 1, 2, 3, 4 ] );
<An object in FinSets>
gap> G_f := [ [ 1, 3 ], [ 2, 4 ], [ 3, 4 ], [ 4, 2 ], [ 5, 4 ] ];
gap> f := MapOfFinSets( S, G_f, T );
<A morphism in FinSets>
gap> G_g := [ [ 1, 3 ], [ 2, 3 ], [ 3, 4 ], [ 4, 2 ], [ 5, 4 ] ];
gap> g := MapOfFinSets( S, G_g, T );
<A morphism in FinSets>
gap> D := [ f, g ];
[ <A morphism in FinSets>, <A morphism in FinSets> ]
gap> C := Coequalizer( D );
<An object in FinSets>
gap> AsList( C );
[ [ 1 ], [ 2 ], [ 3, 4 ] ]
gap> pi := ProjectionOntoCoequalizer( D );
<An epimorphism in FinSets>
gap> AsList( pi );
[ [ 1, [ 1 ] ], [ 2, [ 2 ] ], [ 3, [ 3, 4 ] ], [ 4, [ 3, 4 ] ] ]
gap> mu := MorphismFromSourceToCoequalizer( D );
gap> PreCompose( f, pi ) = mu;
true
gap> G_tau := [ [ 1, 2 ], [ 2, 1 ], [ 3, 2 ], [ 4, 2 ] ];
gap> tau := MapOfFinSets( T, G_tau, FinSet( [ 1, 2 ] ) );
<A morphism in FinSets>
gap> phi := UniversalMorphismFromCoequalizer( D, tau );
<A morphism in FinSets>
gap> AsList( phi );

```

```

[ [ [ 1 ], 2 ], [ [ 2 ], 1 ], [ [ 3, 4 ], 2 ] ]
gap> PreCompose( pi, phi ) = tau;
true
gap> S := FinSet( [ 1, 2, 3, 4, 5 ] );
<An object in FinSets>
gap> T := FinSet( [ 1, 2, 3, 4 ] );
<An object in FinSets>
gap> G_f := [ [ 1, 2 ], [ 2, 3 ], [ 3, 3 ], [ 4, 2 ], [ 5, 4 ] ];
gap> f := MapOfFinSets( S, G_f, T );
<A morphism in FinSets>
gap> G_g := [ [ 1, 2 ], [ 2, 3 ], [ 3, 2 ], [ 4, 2 ], [ 5, 4 ] ];
gap> g := MapOfFinSets( S, G_g, T );
<A morphism in FinSets>
gap> D := [ f, g ];
[ <A morphism in FinSets>, <A morphism in FinSets> ]
gap> C := Coequalizer( D );
<An object in FinSets>
gap> AsList( C );
[ [ 1 ], [ 2, 3 ], [ 4 ] ]
gap> pi := ProjectionOntoCoequalizer( D );
<An epimorphism in FinSets>
gap> AsList( pi );
[ [ 1, [ 1 ] ], [ 2, [ 2, 3 ] ], [ 3, [ 2, 3 ] ], [ 4, [ 4 ] ] ]
gap> PreCompose( f, pi ) = PreCompose( g, pi );
true
gap> mu := MorphismFromSourceToCoequalizer( D );
gap> PreCompose( f, pi ) = mu;
true
gap> G_tau := [ [ 1, 1 ], [ 2, 2 ], [ 3, 2 ], [ 4, 1 ] ];
gap> tau := MapOfFinSets( T, G_tau, FinSet( [ 1, 2 ] ) );
<A morphism in FinSets>
gap> phi := UniversalMorphismFromCoequalizer( D, tau );
<A morphism in FinSets>
gap> AsList( phi );
[ [ [ 1 ], 1 ], [ [ 2, 3 ], 2 ], [ [ 4 ], 1 ] ]
gap> PreCompose( pi, phi ) = tau;
true
gap> A := FinSet( [ "A" ] );
<An object in FinSets>
gap> B := FinSet( [ "B" ] );
<An object in FinSets>
gap> M := FinSetNC( [ A, B ] );
<An object in FinSets>
gap> f := MapOfFinSetsNC( M, [ [ A, A ], [ B, A ] ], M );
<A morphism in FinSets>
gap> g := IdentityMorphism( M );
<An identity morphism in FinSets>
gap> C := Coequalizer( [ f, g ] );
<An object in FinSets>
gap> Length( C );
1
gap> Length( AsList( C )[ 1 ] );

```

```

2
gap> Display( AsList( C )[ 1 ][ 1 ] );
[ "A" ]
gap> Display( AsList( C )[ 1 ][ 2 ] );
[ "B" ]

```

### 1.5.13 Pushout

Example

```

gap> M := FinSet( [ 1 .. 5 ] );
<An object in FinSets>
gap> N1 := FinSet( [ 1, 2, 4 ] );
<An object in FinSets>
gap> iota1 := EmbeddingOfFinSets( N1, M );
<A monomorphism in FinSets>
gap> N2 := FinSet( [ 2, 3 ] );
<An object in FinSets>
gap> iota2 := EmbeddingOfFinSets( N2, M );
<A monomorphism in FinSets>
gap> D := [ iota1, iota2 ];
[ <A monomorphism in FinSets>, <A monomorphism in FinSets> ]
gap> int := FiberProduct( D );
<An object in FinSets>
gap> Display( int );
[ [ 2, 2 ] ]
gap> pi1 := ProjectionInFactorOfFiberProduct( D, 1 );
<A monomorphism in FinSets>
gap> pi2 := ProjectionInFactorOfFiberProduct( D, 2 );
<A monomorphism in FinSets>
gap> UU := Pushout( pi1, pi2 );
<An object in FinSets>
gap> Display( UU );
[ [ [ 1, 1 ] ], [ [ 1, 2 ] ], [ [ 2, 2 ] ], [ [ 1, 4 ] ], [ [ 2, 3 ] ] ]
gap> iota := UniversalMorphismFromPushout( [ pi1, pi2 ], [ iota1, iota2 ] );
<A morphism in FinSets>
gap> U := ImageObject( iota );
<An object in FinSets>
gap> Display( U );
[ 1, 2, 4, 3 ]
gap> UnionOfFinSets( [ N1, N2 ] ) = U;
true

```

### 1.5.14 Cartesian lambda introduction

Example

```

gap> S := FinSet( [ 1 .. 3 ] );
<An object in FinSets>
gap> R := FinSet( [ 1 .. 2 ] );
<An object in FinSets>
gap> f := MapOfFinSets( S, [ [1,2],[2,2],[3,1] ], R );
<A morphism in FinSets>
gap> IsWellDefined( f );
true

```

```

gap> T := TerminalObject( f );
<An object in FinSets>
gap> IsTerminal( T );
true
gap> lf := CartesianLambdaIntroduction( f );
<A split monomorphism in FinSets>
gap> Source( lf ) = T;
true
gap> Length( Range( lf ) );
8
gap> lf( T[1] ) = f;
true

```

### 1.5.15 Topos properties

Example

```

gap> M := FinSet( [ 1 .. 5 ] );
gap> N := FinSet( [ 1, 2, 4 ] );
gap> P := FinSet( [ 1, 4, 8, 9 ] );
gap> G_f := [ [ 1, 1 ], [ 2, 2 ], [ 3, 1 ], [ 4, 2 ], [ 5, 4 ] ];
gap> f := MapOfFinSets( M, G_f, N );
gap> IsWellDefined( f );
true
gap> G_g := [ [ 1, 4 ], [ 2, 4 ], [ 3, 2 ], [ 4, 2 ], [ 5, 1 ] ];
gap> g := MapOfFinSets( M, G_g, N );
gap> IsWellDefined( g );
true
gap> DirectProduct( M, N );
gap> DirectProductOnMorphisms( f, g );
gap> CartesianAssociatorLeftToRight( M, N, P );
gap> CartesianAssociatorRightToLeft( M, N, P );
gap> TerminalObject( FinSets );
gap> CartesianLeftUnitor( M );
gap> CartesianLeftUnitorInverse( M );
gap> CartesianRightUnitor( M );
gap> CartesianRightUnitorInverse( M );
gap> CartesianBraiding( M, N );
gap> CartesianBraidingInverse( M, N );
gap> ExponentialOnObjects( M, N );
gap> ExponentialOnMorphisms( f, g );
gap> CartesianEvaluationMorphism( M, N );
gap> CartesianCoevaluationMorphism( M, N );
gap> DirectProductToExponentialAdjunctionMap( M, N,
>   UniversalMorphismIntoTerminalObject( DirectProduct( M, N ) )
> );
gap> ExponentialToDirectProductAdjunctionMap( M, N,
>   UniversalMorphismFromInitialObject( ExponentialOnObjects( M, N ) )
> );
gap> M := FinSet( [ 1, 2 ] );
gap> N := FinSet( [ "a", "b" ] );
gap> P := FinSet( [ "*" ] );
gap> Q := FinSet( [ "§", "&" ] );
gap> CartesianPreComposeMorphism( M, N, P );

```

```
gap> CartesianPostComposeMorphism( M, N, P );;
gap> DirectProductExponentialCompatibilityMorphism( M, N, P, Q );;
```

### 1.5.16 Subobject Classifier

Example

```
gap> S := FinSet([1,2,3,4,5]);
<An object in FinSets>
gap> A := FinSet([1,5]);
<An object in FinSets>
gap> m := MapOfFinSets(A, List(AsList(A), x -> [x,x]), S);
<A morphism in FinSets>
gap> Display(TruthMorphismIntoSubobjectClassifier(FinSets));
[ [ "*" ], [ [ "*", "true" ] ], [ "true", "false" ] ]
gap> Display(ClassifyingMorphismOfSubobject(m));
[ [ 1, 2, 3, 4, 5 ], [ [ 1, "true" ], [ 2, "false" ], [ 3, "false" ],
[ 4, "false" ], [ 5, "true" ] ], [ "true", "false" ] ]
```

## Chapter 2

# The category of skeletal finite sets

### 2.1 Skeletal GAP Categories

#### 2.1.1 IsCategoryOfSkeletalFinSets (for IsCapCategory)

▷ `IsCategoryOfSkeletalFinSets(object)` (filter)  
**Returns:** true or false  
The GAP category of categories of skeletal finite sets.

#### 2.1.2 IsSkeletalFiniteSet (for IsCapCategoryObject and IsCellOfSkeletalCategory)

▷ `IsSkeletalFiniteSet(object)` (filter)  
**Returns:** true or false  
The GAP category of objects in the category of skeletal finite sets.

#### 2.1.3 IsSkeletalFiniteSetMap (for IsCapCategoryMorphism and IsCellOfSkeletalCategory)

▷ `IsSkeletalFiniteSetMap(object)` (filter)  
**Returns:** true or false  
The GAP category of morphisms in the category of skeletal finite sets.

### 2.2 Skeletal Attributes

#### 2.2.1 Length (for IsSkeletalFiniteSet)

▷ `Length(M)` (attribute)  
**Returns:** an integer  
The integer defining the skeletal finite set  $M$ , i.e., `Length( FinSet( n ) ) = n`.

#### 2.2.2 AsList (for IsSkeletalFiniteSet)

▷ `AsList(M)` (attribute)  
**Returns:** a list  
The list associated to a skeletal finite set, i.e., `AsList( FinSet( n ) ) = [ 1 .. n ]`.

## 2.3 Skeletal Constructors

### 2.3.1 CategoryOfSkeletalFinSets

▷ `CategoryOfSkeletalFinSets()` (operation)

**Returns:** a CAP category

Construct a category of skeletal finite sets. Accepts the options `overhead` (default: `true`) and `FinalizeCategory` (default: `true`).

### 2.3.2 SkeletalFinSets

▷ `SkeletalFinSets` (global variable)

The default instance of the category of skeletal finite sets. It is automatically created while loading this package.

### 2.3.3 FinSet (for IsInt)

▷ `FinSet(n)` (operation)

**Returns:** a CAP object

Construct a skeletal finite set residing in the default instance of the category of skeletal finite sets `SkeletalFinSets` of order given by the nonnegative integer  $n$ .

Example

```
gap> m := FinSet( 7 );
<An object in SkeletalFinSets>
gap> IsWellDefined( m );
true
gap> n := FinSet( -2 );
<An object in SkeletalFinSets>
gap> IsWellDefined( n );
false
gap> n := FinSet( 3 );
<An object in SkeletalFinSets>
gap> IsWellDefined( n );
true
gap> p := FinSet( 4 );
<An object in SkeletalFinSets>
gap> IsWellDefined( p );
true
```

### 2.3.4 FinSet (for IsCategoryOfSkeletalFinSets, IsInt)

▷ `FinSet(C, n)` (operation)

**Returns:** a CAP object

Construct a skeletal finite set residing in the given category of skeletal finite sets  $C$  of order given by the nonnegative integer  $n$ .



### 2.3.5 MapOfFinSets (for IsSkeletalFiniteSet, IsList, IsSkeletalFiniteSet)

▷ MapOfFinSets( $s$ ,  $G$ ,  $t$ ) (operation)

**Returns:** a CAP morphism

Construct a map  $\phi : s \rightarrow t$  of the skeletal finite sets  $s$  and  $t$ , i.e., a morphism in the CAP category of  $s$ , where  $G$  is a list of integers in  $t$  describing the graph of  $\phi$ .

Example

```
gap> s := FinSet( 3 );
<An object in SkeletalFinSets>
gap> t := FinSet( 7 );
<An object in SkeletalFinSets>
gap> phi := MapOfFinSets( s, [7, 5, 5], t );
<A morphism in SkeletalFinSets>
gap> IsWellDefined( phi );
true
gap> Display( phi );
[ 3, [ 7, 5, 5 ], 7 ]
```

## 2.4 Skeletal Tools

### 2.4.1 ListOp (for IsSkeletalFiniteSet, IsFunction)

▷ ListOp( $s$ ,  $f$ ) (operation)

**Returns:** a list

Returns List( AsList(  $s$  ),  $f$  ).

### 2.4.2 EmbeddingOfFinSets (for IsSkeletalFiniteSet, IsSkeletalFiniteSet)

▷ EmbeddingOfFinSets( $s$ ,  $t$ ) (operation)

**Returns:** a CAP morphism

Construct the embedding  $\iota : s \rightarrow t$  of the finite sets  $s$  and  $t$ , where  $s$  must be subset of  $t$ .

### 2.4.3 Preimage (for IsSkeletalFiniteSetMap, IsList)

▷ Preimage( $\phi$ ,  $t$ ) (operation)

**Returns:** a CAP object

Compute the Preimage of  $t$  under the morphism  $\phi$ .

### 2.4.4 ImageObject (for IsSkeletalFiniteSetMap, IsSkeletalFiniteSet)

▷ ImageObject( $\phi$ ,  $s_*$ ) (operation)

**Returns:** a CAP object

Compute the image of  $s_*$  under the morphism  $\phi$ .

### 2.4.5 CallFuncList (for IsSkeletalFiniteSetMap, IsList)

▷ CallFuncList( $\phi$ ,  $L$ ) (operation)

**Returns:** a list

Returns the image of  $L[1]$  under the map  $\phi$  assuming  $L[1]$  is a positive integer smaller or equal to  $\text{Length}(\text{Source}(\phi))$ .

## 2.5 Skeletal Examples

### 2.5.1 SkeletalIsHomSetInhabited

Example

```
gap> L := FinSet( 0 );
<An object in SkeletalFinSets>
gap> M := FinSet( 2 );
<An object in SkeletalFinSets>
gap> N := FinSet( 3 );
<An object in SkeletalFinSets>
gap> IsHomSetInhabited( L, L );
true
gap> IsHomSetInhabited( M, L );
false
gap> IsHomSetInhabited( L, M );
true
gap> IsHomSetInhabited( M, N );
true
```

### 2.5.2 SkeletalWellDefined

Example

```
gap> s := FinSet( 7 );
<An object in SkeletalFinSets>
gap> t := FinSet( 4 );
<An object in SkeletalFinSets>
gap> psi := MapOfFinSets( s, [ 1, 3, 2, 3, 2, 4 ], t );
<A morphism in SkeletalFinSets>
gap> IsWellDefined( psi );
false
gap> psi := MapOfFinSets( s, [ 1, 3, 2, 3, 2, 4, -1 ], t );
<A morphism in SkeletalFinSets>
gap> IsWellDefined( psi );
false
gap> psi := MapOfFinSets( s, [ 2, 3, 2, 5, 3, 2, 4 ], t );
<A morphism in SkeletalFinSets>
gap> IsWellDefined( psi );
false
gap> psi := MapOfFinSets( s, [ 1, 3, 2, 4, 3, 2, 4 ], t );
<A morphism in SkeletalFinSets>
gap> IsWellDefined( psi );
true
```

### 2.5.3 SkeletalPreCompose

Example

```
gap> m := FinSet( 3 );
<An object in SkeletalFinSets>
gap> n := FinSet( 5 );
```

```

<An object in SkeletalFinSets>
gap> p := FinSet( 7 );
<An object in SkeletalFinSets>
gap> psi := MapOfFinSets( m, [ 2, 5, 3 ], n );
<A morphism in SkeletalFinSets>
gap> phi := MapOfFinSets( n, [ 1, 4, 6, 6, 3 ], p );
<A morphism in SkeletalFinSets>
gap> alpha := PreCompose( psi, phi );
<A morphism in SkeletalFinSets>
gap> Display( alpha );
[ 3, [ 4, 3, 6 ], 7 ]

```

### 2.5.4 Skeletal Monomorphisms and Epimorphisms

Example

```

gap> m := FinSet( 3 );
<An object in SkeletalFinSets>
gap> n := FinSet( 5 );
<An object in SkeletalFinSets>
gap> p := FinSet( 7 );
<An object in SkeletalFinSets>
gap> psi := MapOfFinSets( m, [ 1, 3, 5 ], n );
<A morphism in SkeletalFinSets>
gap> IsEpimorphism( psi );
false
gap> IsSplitEpimorphism( psi );
false
gap> IsMonomorphism( psi );
true
gap> IsSplitMonomorphism( psi );
true
gap> psi := MapOfFinSets( p, [ 1, 3, 2, 3, 3, 2, 1 ], m );
<A morphism in SkeletalFinSets>
gap> IsEpimorphism( psi );
true
gap> IsSplitEpimorphism( psi );
true
gap> IsMonomorphism( psi );
false
gap> IsSplitMonomorphism( psi );
false

```

### 2.5.5 Skeletal Initial and Terminal Objects

Example

```

gap> m := FinSet( 8 );
<An object in SkeletalFinSets>
gap> IsInitial( m );
false
gap> IsTerminal( m );
false
gap> i := InitialObject( m );
<An object in SkeletalFinSets>

```

```

gap> IsInitial( i );
true
gap> IsTerminal( i );
false
gap> iota := UniversalMorphismFromInitialObject( m );
<A morphism in SkeletalFinSets>
gap> AsList( i );
[ ]
gap> t := TerminalObject( m );
<An object in SkeletalFinSets>
gap> AsList( t );
[ 1 ]
gap> IsInitial( t );
false
gap> IsTerminal( t );
true
gap> pi := UniversalMorphismIntoTerminalObject( m );
<A morphism in SkeletalFinSets>
gap> IsIdenticalObj( Range( pi ), t );
true
gap> pi_t := UniversalMorphismIntoTerminalObjectWithGivenTerminalObject( m, t );
<A morphism in SkeletalFinSets>
gap> AsList( pi_t );
[ 1, 1, 1, 1, 1, 1, 1, 1 ]
gap> pi = pi_t;
true

```

### 2.5.6 Projective and Injective Objects

Example

```

gap> I := FinSet( 0 );
<An object in SkeletalFinSets>
gap> T := FinSet( 1 );
<An object in SkeletalFinSets>
gap> M := FinSet( 2 );
<An object in SkeletalFinSets>
gap> IsProjective( I );
true
gap> IsProjective( T );
true
gap> IsProjective( M );
true
gap> IsOne( EpimorphismFromSomeProjectiveObject( I ) );
true
gap> IsOne( EpimorphismFromSomeProjectiveObject( M ) );
true

```

Example

```

gap> I := FinSet( 0 );
<An object in SkeletalFinSets>
gap> T := FinSet( 1 );
<An object in SkeletalFinSets>
gap> M := FinSet( 2 );

```

```

<An object in SkeletalFinSets>
gap> IsInjective( I );
false
gap> IsInjective( T );
true
gap> IsInjective( M );
true
gap> IsIsomorphism( MonomorphismIntoSomeInjectiveObject( I ) );
false
gap> IsMonomorphism( MonomorphismIntoSomeInjectiveObject( I ) );
true
gap> IsOne( MonomorphismIntoSomeInjectiveObject( M ) );
true

```

## 2.5.7 Skeletal Product

Example

```

gap> m := FinSet( 7 );
<An object in SkeletalFinSets>
gap> n := FinSet( 3 );
<An object in SkeletalFinSets>
gap> p := FinSet( 4 );
<An object in SkeletalFinSets>
gap> d := DirectProduct( [ m, n, p ] );
<An object in SkeletalFinSets>
gap> AsList( d );
[ 1 .. 84 ]
gap> pi1 := ProjectionInFactorOfDirectProduct( [ m, n, p ], 1 );
<A morphism in SkeletalFinSets>
gap> Display( pi1 );
[ 84,
  [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
    2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4,
    4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6,
    6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7 ], 7
]
gap> pi2 := ProjectionInFactorOfDirectProduct( [ m, n, p ], 2 );
<A morphism in SkeletalFinSets>
gap> Display( pi2 );
[ 84,
  [ 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3,
    3, 3, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 1, 1, 1, 1, 2, 2, 2,
    2, 3, 3, 3, 3, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 1, 1, 1, 1,
    2, 2, 2, 2, 3, 3, 3, 3, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3 ], 3
]
gap> pi3 := ProjectionInFactorOfDirectProduct( [ m, n, p ], 3 );
<A morphism in SkeletalFinSets>
gap> Display( pi3 );
[ 84,
  [ 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2,
    3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3,
    4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4,
    1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4 ], 4
]

```

```

]
gap> psi := UniversalMorphismIntoDirectProduct( [ m, n, p ], [ pi1, pi2, pi3 ] );
<A morphism in SkeletalFinSets>
gap> psi = IdentityMorphism( d );
true

```

### 2.5.8 Skeletal Coproduct

Example

```

gap> m := FinSet( 7 );
<An object in SkeletalFinSets>
gap> n := FinSet( 3 );
<An object in SkeletalFinSets>
gap> p := FinSet( 4 );
<An object in SkeletalFinSets>
gap> c := Coproduct( m, n, p );
<An object in SkeletalFinSets>
gap> AsList( c );
[ 1 .. 14 ]
gap> iota1 := InjectionOfCofactorOfCoproduct( [ m, n, p ], 1 );
<A morphism in SkeletalFinSets>
gap> IsWellDefined( iota1 );
true
gap> Display( iota1 );
[ 7, [ 1, 2, 3, 4, 5, 6, 7 ], 14 ]
gap> iota2 := InjectionOfCofactorOfCoproduct( [ m, n, p ], 2 );
<A morphism in SkeletalFinSets>
gap> IsWellDefined( iota2 );
true
gap> Display( iota2 );
[ 3, [ 8, 9, 10 ], 14 ]
gap> iota3 := InjectionOfCofactorOfCoproduct( [ m, n, p ], 3 );
<A morphism in SkeletalFinSets>
gap> IsWellDefined( iota3 );
true
gap> Display( iota3 );
[ 4, [ 11, 12, 13, 14 ], 14 ]
gap> psi := UniversalMorphismFromCoproduct( [ m, n, p ],
>                                           [ iota1, iota2, iota3 ]
>                                           );
<A morphism in SkeletalFinSets>
gap> psi = IdentityMorphism( Coproduct( [ m, n, p ] ) );
true

```

### 2.5.9 Skeletal Image

Example

```

gap> m := FinSet( 7 );
<An object in SkeletalFinSets>
gap> n := FinSet( 3 );
<An object in SkeletalFinSets>
gap> phi := MapOfFinSets( n, [7, 5, 5], m );
<A morphism in SkeletalFinSets>

```

```

gap> IsWellDefined( phi );
true
gap> ImageObject( phi );
<An object in SkeletalFinSets>
gap> Length( ImageObject( phi ) );
2
gap> s := FinSet( 2 );
<An object in SkeletalFinSets>
gap> I := ImageObject( phi, s );
<An object in SkeletalFinSets>
gap> Length( I );
2

```

Example

```

gap> m := FinSet( 7 );
<An object in SkeletalFinSets>
gap> n := FinSet( 3 );
<An object in SkeletalFinSets>
gap> phi := MapOfFinSets( n, [ 7, 5, 5 ], m );
<A morphism in SkeletalFinSets>
gap> pi := ImageEmbedding( phi );
<A monomorphism in SkeletalFinSets>
gap> Display( pi );
[ 2, [ 5, 7 ], 7 ]

```

Example

```

gap> m := FinSet( 7 );
<An object in SkeletalFinSets>
gap> n := FinSet( 3 );
<An object in SkeletalFinSets>
gap> phi := MapOfFinSets( n, [ 7, 5, 5 ], m );
<A morphism in SkeletalFinSets>
gap> IsWellDefined( phi );
true
gap> f := CostrictionToImage( phi );
<An epimorphism in SkeletalFinSets>
gap> Display( f );
[ 3, [ 2, 1, 1 ], 2 ]
gap> IsWellDefined( f );
true
gap> IsEpimorphism( f );
true
gap> IsSplitEpimorphism( f );
true
gap> m := FinSet( 77 );
<An object in SkeletalFinSets>
gap> n := FinSet( 4 );
<An object in SkeletalFinSets>
gap> phi := MapOfFinSets( n, [ 77, 2, 25, 2 ], m );
<A morphism in SkeletalFinSets>
gap> IsWellDefined( phi );
true
gap> iota := ImageEmbedding( phi );
<A monomorphism in SkeletalFinSets>

```

```

gap> pi := CostrictionToImage( phi );
<An epimorphism in SkeletalFinSets>
gap> Display( pi );
[ 4, [ 3, 1, 2, 1 ], 3 ]
gap> IsWellDefined( pi );
true
gap> IsEpimorphism( pi );
true
gap> IsSplitEpimorphism( pi );
true
gap> PreCompose( pi, iota ) = phi;
true

```

### 2.5.10 Skeletal Preimage

Example

```

gap> m := FinSet( 7 );
<An object in SkeletalFinSets>
gap> n := FinSet( 3 );
<An object in SkeletalFinSets>
gap> phi := MapOfFinSets( n, [7, 5, 5], m );
<A morphism in SkeletalFinSets>
gap> P := Preimage( phi, [2] );
[ ]
gap> P := Preimage( phi, AsList( FinSet( 5 ) ) );
[ 2, 3 ]

```

### 2.5.11 Skeletal Equalizer

Example

```

gap> S := FinSet( 5 );
<An object in SkeletalFinSets>
gap> T := FinSet( 3 );
<An object in SkeletalFinSets>
gap> f1 := MapOfFinSets( S, [ 3, 3, 1, 2, 2 ], T );
<A morphism in SkeletalFinSets>
gap> f2 := MapOfFinSets( S, [ 3, 2, 3, 1, 2 ], T );
<A morphism in SkeletalFinSets>
gap> f3 := MapOfFinSets( S, [ 3, 1, 2, 1, 2 ], T );
<A morphism in SkeletalFinSets>
gap> D := [ f1, f2, f3 ];
gap> Eq := Equalizer( D );
<An object in SkeletalFinSets>
gap> Length( Eq );
2
gap> iota := EmbeddingOfEqualizer( D );
<A monomorphism in SkeletalFinSets>
gap> Display( iota );
[ 2, [ 1, 5 ], 5 ]
gap> phi := MapOfFinSets( FinSet( 2 ), [ 5, 1 ], S );
gap> IsWellDefined( phi );
true
gap> psi := UniversalMorphismIntoEqualizer( D, phi );

```



```

<A morphism in SkeletalFinSets>
gap> IsWellDefined( psi );
true
gap> Display( psi );
[ 2, [ 2, 1 ], 2 ]
gap> PreCompose( psi, iota ) = phi;
true
gap> D := [ f2, f3 ];
[ <A morphism in SkeletalFinSets>, <A morphism in SkeletalFinSets> ]
gap> Eq := Equalizer( D );
<An object in SkeletalFinSets>
gap> Length( Eq );
3
gap> psi := EmbeddingOfEqualizer( D );
<A monomorphism in SkeletalFinSets>
gap> Display( psi );
[ 3, [ 1, 4, 5 ], 5 ]

```

### 2.5.12 Skeletal Pullback

Example

```

gap> m := FinSet( 5 );
<An object in SkeletalFinSets>
gap> n1 := FinSet( 3 );
<An object in SkeletalFinSets>
gap> iota1 := EmbeddingOfFinSets( n1, m );
<A monomorphism in SkeletalFinSets>
gap> Display( iota1 );
[ 3, [ 1, 2, 3 ], 5 ]
gap> n2 := FinSet( 4 );
<An object in SkeletalFinSets>
gap> iota2 := EmbeddingOfFinSets( n2, m );
<A monomorphism in SkeletalFinSets>
gap> Display( iota2 );
[ 4, [ 1, 2, 3, 4 ], 5 ]
gap> D := [ iota1, iota2 ];
[ <A monomorphism in SkeletalFinSets>, <A monomorphism in SkeletalFinSets> ]
gap> Fib := FiberProduct( D );
<An object in SkeletalFinSets>
gap> Display( Fib );
3
gap> pi1 := ProjectionInFactorOfFiberProduct( D, 1 );
<A monomorphism in SkeletalFinSets>
gap> Display( pi1 );
[ 3, [ 1, 2, 3 ], 3 ]
gap> int1 := ImageObject( pi1 );
<An object in SkeletalFinSets>
gap> Display( int1 );
3
gap> pi2 := ProjectionInFactorOfFiberProduct( D, 2 );
<A monomorphism in SkeletalFinSets>
gap> Display( pi2 );
[ 3, [ 1, 2, 3 ], 4 ]

```

```

gap> int2 := ImageObject( pi2 );
<An object in SkeletalFinSets>
gap> Display( int2 );
3
gap> omega1 := PreCompose( pi1, iota1 );
<A monomorphism in SkeletalFinSets>
gap> omega2 := PreCompose( pi2, iota2 );
<A monomorphism in SkeletalFinSets>
gap> omega1 = omega2;
true
gap> Display( omega1 );
[ 3, [ 1, 2, 3 ], 5 ]

```

### 2.5.13 Skeletal Coequalizer

Example

```

gap> s := FinSet( 5 );
<An object in SkeletalFinSets>
gap> t := FinSet( 4 );
<An object in SkeletalFinSets>
gap> f := MapOfFinSets( s, [ 3, 4, 4, 2, 4 ], t );
<A morphism in SkeletalFinSets>
gap> g := MapOfFinSets( s, [ 3, 3, 4, 2, 4 ], t );
<A morphism in SkeletalFinSets>
gap> D := [ f, g ];
[ <A morphism in SkeletalFinSets>, <A morphism in SkeletalFinSets> ]
gap> C := Coequalizer( D );
<An object in SkeletalFinSets>
gap> Length( C );
3
gap> pi := ProjectionOntoCoequalizer(D);
<An epimorphism in SkeletalFinSets>
gap> Display( pi );
[ 4, [ 1, 2, 3, 3 ], 3 ]
gap> tau := MapOfFinSets( t, [2, 1, 2, 2], FinSet( 2 ) );
<A morphism in SkeletalFinSets>
gap> phi := UniversalMorphismFromCoequalizer( D, tau );
<A morphism in SkeletalFinSets>
gap> Display( phi );
[ 3, [ 2, 1, 2 ], 2 ]
gap> PreCompose( pi, phi ) = tau;
true
gap> s := FinSet( 5 );
<An object in SkeletalFinSets>
gap> t := FinSet( 4 );
<An object in SkeletalFinSets>
gap> f := MapOfFinSets( s, [ 2, 3, 3, 2, 4 ], t );
<A morphism in SkeletalFinSets>
gap> g := MapOfFinSets( s, [ 2, 3, 2, 2, 4 ], t );
<A morphism in SkeletalFinSets>
gap> D := [ f, g ];
[ <A morphism in SkeletalFinSets>, <A morphism in SkeletalFinSets> ]
gap> C := Coequalizer( D );

```

```

<An object in SkeletalFinSets>
gap> Length( C );
3
gap> pi := ProjectionOntoCoequalizer( D );
<An epimorphism in SkeletalFinSets>
gap> Display( pi );
[ 4, [ 1, 2, 2, 3 ], 3 ]
gap> PreCompose( f, pi ) = PreCompose( g, pi );
true
gap> tau := MapOfFinSets( t, [1, 2, 2, 1 ], FinSet( 2 ) );
<A morphism in SkeletalFinSets>
gap> phi := UniversalMorphismFromCoequalizer( D, tau );
<A morphism in SkeletalFinSets>
gap> Display( phi );
[ 3, [ 1, 2, 1 ], 2 ]
gap> PreCompose( pi, phi ) = tau;
true
gap> s := FinSet( 2 );;
gap> t := FinSet( 3 );;
gap> f := MapOfFinSets( s, [ 1, 2 ], t );;
gap> IsWellDefined( f );
true
gap> g := MapOfFinSets( s, [ 2, 3 ], t );;
gap> IsWellDefined( g );
true
gap> C := Coequalizer( [ f, g ] );
<An object in SkeletalFinSets>
gap> Length( C );
1

```

### 2.5.14 Skeletal Pushout

Example

```

gap> M := FinSet( 5 );
<An object in SkeletalFinSets>
gap> N1 := FinSet( 3 );
<An object in SkeletalFinSets>
gap> iota1 := EmbeddingOfFinSets( N1, M );
<A monomorphism in SkeletalFinSets>
gap> Display( iota1 );
[ 3, [ 1, 2, 3 ], 5 ]
gap> N2 := FinSet( 2 );
<An object in SkeletalFinSets>
gap> iota2 := EmbeddingOfFinSets( N2, M );
<A monomorphism in SkeletalFinSets>
gap> Display( iota2 );
[ 2, [ 1, 2 ], 5 ]
gap> D := [ iota1, iota2 ];
[ <A monomorphism in SkeletalFinSets>, <A monomorphism in SkeletalFinSets> ]
gap> Fib := FiberProduct( D );
<An object in SkeletalFinSets>
gap> Display( Fib );
2

```

```

gap> pi1 := ProjectionInFactorOfFiberProduct( D, 1 );
<A monomorphism in SkeletalFinSets>
gap> Display( pi1 );
[ 2, [ 1, 2 ], 3 ]
gap> pi2 := ProjectionInFactorOfFiberProduct( D, 2 );
<A monomorphism in SkeletalFinSets>
gap> Display( pi2 );
[ 2, [ 1, 2 ], 2 ]
gap> ## The easy way
>
> D := [ pi1, pi2 ];
[ <A monomorphism in SkeletalFinSets>, <A monomorphism in SkeletalFinSets> ]
gap> UU := Pushout( D );
<An object in SkeletalFinSets>
gap> Display( UU );
3
gap> kappa1 := InjectionOfCofactorOfPushout( D, 1 );
<A morphism in SkeletalFinSets>
gap> Display( kappa1 );
[ 3, [ 1, 2, 3 ], 3 ]
gap> kappa2 := InjectionOfCofactorOfPushout( D, 2 );
<A morphism in SkeletalFinSets>
gap> Display( kappa2 );
[ 2, [ 1, 2 ], 3 ]
gap> PreCompose( pi1, kappa1 ) = PreCompose( pi2, kappa2 );
true
gap> ## The long way
>
> Co := Coproduct( N1, N2 );
<An object in SkeletalFinSets>
gap> Display( Co );
5
gap> iota_1 := InjectionOfCofactorOfCoproduct( [ N1, N2 ], 1 );
<A morphism in SkeletalFinSets>
gap> Display( iota_1 );
[ 3, [ 1, 2, 3 ], 5 ]
gap> iota_2 := InjectionOfCofactorOfCoproduct( [ N1, N2 ], 2 );
<A morphism in SkeletalFinSets>
gap> Display( iota_2 );
[ 2, [ 4, 5 ], 5 ]
gap> alpha := PreCompose( pi1, iota_1 );
<A morphism in SkeletalFinSets>
gap> Display( alpha );
[ 2, [ 1, 2 ], 5 ]
gap> beta := PreCompose( pi2, iota_2 );
<A morphism in SkeletalFinSets>
gap> Display( beta );
[ 2, [ 4, 5 ], 5 ]
gap> Cq := Coequalizer( [ alpha, beta ] );
<An object in SkeletalFinSets>
gap> Display( Cq );
3

```

```

gap> psi := ProjectionOntoCoequalizer( [ alpha, beta ] );
<An epimorphism in SkeletalFinSets>
gap> Display( psi );
[ 5, [ 1, 2, 3, 1, 2 ], 3 ]
gap> Display( PreCompose( iota_1, psi ) );
[ 3, [ 1, 2, 3 ], 3 ]
gap> Display( PreCompose( iota_2, psi ) );
[ 2, [ 1, 2 ], 3 ]
gap> PreCompose( alpha, psi ) = PreCompose( beta, psi );
true

```

### 2.5.15 Skeletal Lift

Example

```

gap> m := FinSet( 5 );
<An object in SkeletalFinSets>
gap> n := FinSet( 4 );
<An object in SkeletalFinSets>
gap> f := MapOfFinSets( m, [ 2, 2, 1, 1, 3 ], n );
<A morphism in SkeletalFinSets>
gap> g := MapOfFinSets( m, [ 5, 5, 4, 4, 5 ], m );
<A morphism in SkeletalFinSets>
gap> IsColiftable( f, g );
true
gap> chi := Colift( f, g );
<A morphism in SkeletalFinSets>
gap> Display( chi );
[ 4, [ 4, 5, 5, 1 ], 5 ]
gap> PreCompose( f, Colift( f, g ) ) = g;
true
gap> IsColiftable( g, f );
false
gap> Colift( g, f );
fail

```

Example

```

gap> m := FinSet( 3 );
<An object in SkeletalFinSets>
gap> n := FinSet( 4 );
<An object in SkeletalFinSets>
gap> f := MapOfFinSets( m, [ 2, 2, 1 ], m );
<A morphism in SkeletalFinSets>
gap> g := MapOfFinSets( n, [ 3, 2, 1, 2 ], m );
<A morphism in SkeletalFinSets>
gap> IsLiftable( f, g );
true
gap> chi := Lift( f, g );
<A morphism in SkeletalFinSets>
gap> Display( chi );
[ 3, [ 2, 2, 3 ], 4 ]
gap> PreCompose( Lift( f, g ), g ) = f;
true
gap> IsLiftable( g, f );
false

```

```

gap> Lift( g, f );
fail
gap> k := FinSet( 100000 );
<An object in SkeletalFinSets>
gap> h := ListWithIdenticalEntries( Length( k ) - 3, 3 );;
gap> h := Concatenation( h, [ 2, 1, 2 ] );;
gap> h := MapOfFinSets( k, h, m );
<A morphism in SkeletalFinSets>
gap> IsLiftable( f, h );
true
gap> IsLiftable( h, f );
false

```

### 2.5.16 Skeletal Colift

### 2.5.17 Skeletal topos properties

Example

```

gap> M := FinSet( 4 );;
gap> N := FinSet( 3 );;
gap> P := FinSet( 4 );;
gap> G_f := [ 1, 2, 1, 3 ];;
gap> f := MapOfFinSets( M, G_f, N );;
gap> IsWellDefined( f );
true
gap> G_g := [ 3, 3, 2, 1 ];;
gap> g := MapOfFinSets( M, G_g, N );;
gap> IsWellDefined( g );
true
gap> DirectProduct( M, N );;
gap> DirectProductOnMorphisms( f, g );;
gap> CartesianAssociatorLeftToRight( M, N, P );;
gap> CartesianAssociatorRightToLeft( M, N, P );;
gap> TerminalObject( FinSets );;
gap> CartesianLeftUnitor( M );;
gap> CartesianLeftUnitorInverse( M );;
gap> CartesianRightUnitor( M );;
gap> CartesianRightUnitorInverse( M );;
gap> CartesianBraiding( M, N );;
gap> CartesianBraidingInverse( M, N );;
gap> ExponentialOnObjects( M, N );;
gap> ExponentialOnMorphisms( f, g );;
gap> CartesianEvaluationMorphism( M, N );;

```

# Index

- `[]`
  - for `IsFiniteSet`, `IsInt`, 8
- `AsList`
  - for `IsFiniteSet`, 3
  - for `IsFiniteSetMap`, 3
  - for `IsSkeletalFiniteSet`, 22
- `\in`
  - for `IsObject`, `IsFiniteSet`, 8
- `CallFuncList`
  - for `IsFiniteSetMap`, `IsList`, 9
  - for `IsSkeletalFiniteSetMap`, `IsList`, 24
- `CategoryOfSkeletalFinSets`, 23
- `EmbeddingOfFinSets`
  - for `IsFiniteSet`, `IsFiniteSet`, 9
  - for `IsSkeletalFiniteSet`, `IsSkeletalFiniteSet`, 24
- `FilteredOp`
  - for `IsFiniteSet`, `IsFunction`, 8
- `FinSet`
  - for `IsCategoryOfSkeletalFinSets`, `IsInt`, 23
  - for `IsInt`, 23
  - for `IsList`, 4
- `FinSetNC`
  - for `IsList`, 4
- `FirstOp`
  - for `IsFiniteSet`, `IsFunction`, 9
- `ImageObject`
  - for `IsFiniteSetMap`, `IsFiniteSet`, 9
  - for `IsSkeletalFiniteSetMap`, `IsSkeletalFiniteSet`, 24
- `IsCategoryOfSkeletalFinSets`
  - for `IsCapCategory`, 22
- `IsEqualForElementsOfFinSets`
  - for `IsObject`, `IsObject`, 7
- `IsFiniteSet`
  - for `IsCapCategoryObject`, 3
- `IsFiniteSetMap`
  - for `IsCapCategoryMorphism`, 3
- `IsSkeletalFiniteSet`
  - for `IsCapCategoryObject` and `IsCellOfSkeletalCategory`, 22
- `IsSkeletalFiniteSetMap`
  - for `IsCapCategoryMorphism` and `IsCellOfSkeletalCategory`, 22
- `Iterator`
  - for `IsFiniteSet`, 8
- `Length`
  - for `IsFiniteSet`, 3
  - for `IsSkeletalFiniteSet`, 22
- `ListOp`
  - for `IsFiniteSet`, `IsFiniteSetMap`, 9
  - for `IsFiniteSet`, `IsFunction`, 8
  - for `IsSkeletalFiniteSet`, `IsFunction`, 24
- `MapOfFinSets`
  - for `IsFiniteSet`, `IsList`, `IsFiniteSet`, 5
  - for `IsSkeletalFiniteSet`, `IsList`, `IsSkeletalFiniteSet`, 24
- `MapOfFinSetsNC`
  - for `IsFiniteSet`, `IsList`, `IsFiniteSet`, 6
- `Preimage`
  - for `IsFiniteSetMap`, `IsFiniteSet`, 9
  - for `IsSkeletalFiniteSetMap`, `IsList`, 24
- `ProjectionOfFinSets`
  - for `IsFiniteSet`, `IsFiniteSet`, 9
- `SkeletalFinSets`, 23
- `UnionOfFinSets`
  - for `IsList`, 8