FinGSetsForCAP

The elementary topos of (skeletal) finite G-sets

2019.03.03

3 March 2019

Mohamed Barakat

Julia Mickisch

Fabian Zickgraf

Mohamed Barakat
Email: 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
Homepage: https://github.com/juliamick/
Address: Walter-Flex-Str. 3
57068 Siegen
Germany

Fabian Zickgraf
Email: fabian.zickgraf@uni-siegen.de
Homepage: https://github.com/zickgraf/
Address: Walter-Flex-Str. 3
57068 Siegen
Germany

Contents

Chapter 1

Mathematical background

1.1 The categories G-Sets and Skeletal-G-Sets

In this section, we first want to define the category **G-Sets** and prove the existence of some category-theoretic constructions which we will need later. Afterwards, we define the category **Skeletal-G-Sets** in a way which mirrors the data structures of our implementation in CAP. Thus, from an abstract point of view, the definitions might seem unusual. However, we will make use of these data structures to give explicit algorithms for some category-theoretic constructions in **Skeletal-G-Sets**. In particular, we are interested in algorithms for finite equalizers and coequalizers because in conjunction with the algorithms for finite products and coproducts, which are explained in [Mic18], all finite limits and colimits are algorithmically accessible. Important notions which we need as a preparation for equalizers and coequalizers are sub-G-sets, mono- and epimorphisms, images and preimages, and lifts and colifts along mono- and epimorphisms respectively.

1.1.1 The category G-Sets

Throughout the whole chapter let G be a finite group, k the number of conjugacy classes of subgroups of G, $\mathscr{I} := \{1,\dots,k\}$ and $\mathscr{U} = \{U_i \mid i \in \mathscr{I}\}$ a set of representatives of the conjugacy classes of subgroups of G with $U_1 = \{1\}$, $U_k = G$ and $|U_i| \leq |U_j|$ if $i < j$. If G is a group named `group` in GAP then we fix $\mathscr{U}$ by setting

$$U_i := \texttt{RepresentativeTom(TableOfMarks(group), i)}.$$

For example, for $G = S_3 = \texttt{SymmetricGroup(3)}$ we have $k = 4$ and

$$\begin{aligned} U_1 &= \{()\}, \\ U_2 &= \langle (23) \rangle \leq S_3, \\ U_3 &= A_3 \leq S_3, \\ U_4 &= S_3. \end{aligned}$$

Definition 1.1.1 (The category **G-Sets**)**.** A *right action* of G on a set Ω is a map $\Omega \times G \to \Omega$, $(\omega, g) \mapsto \omega g$ with the following properties:

- $\omega 1 = \omega$ for all $\omega \in \Omega$ and
- $(\omega g)h = \omega(gh)$ for all $\omega \in \Omega$ and all $g, h \in G$.

A set Ω together with a right action is called a *right G-set*. A *morphism* of right G-sets $\varphi\colon \Omega \to \Delta$ is a *G-equivariant* map of sets, that is, it holds that $\varphi(\omega g) = \varphi(\omega)g$ for all $\omega \in \Omega$ and all $g \in G$. We will never encounter left actions or left G-sets and thus we will simply say *action* instead of *right action* and *G-set* instead of *right G-set* from now on.

We only deal with the case of *finite* G-sets and define the category **G-Sets** as follows: its objects are the finite G-sets and morphisms are morphisms of G-sets. From now on, any G-set will be finite.

Definition 1.1.2 (Orbits)**.** Let Ω be a G-set and let $\omega \in \Omega$. Then the *orbit* of ω is the set $\omega G := \{\omega g \mid g \in G\}$, and Ω is called *transitive* if it consists of a single orbit.

Example 1.1.3 (Right cosets as G-sets)**.** For any subgroup $U \leq G$, the set of right cosets $U\backslash G$ is a G-set by right multiplication. In particular, $G = \{1\}\backslash G$ is a G-set by right multiplication. In the following, sets of right cosets as well as G itself will always be regarded as G-sets by right multiplication.

Definition 1.1.4 (The forgetful functor)**.** We define a functor $U\colon$ **G-Sets** $\to$ **Sets** which sends every G-set to its underlying set and a morphism of G-sets to the set-theoretic map. Since morphisms of G-sets are just maps of sets, this really is a functor. Two morphisms of G-sets are equal if and only if they are equal as maps of sets, so U is faithful. We call U the *forgetful functor* from **G-Sets** to **Sets**.

Lemma 1.1.5 (Representability of the forgetful functor)**.** *The forgetful functor U and the hom-functor* $\mathrm{Hom}_{\textbf{G-Sets}}(G,-)$ *are isomorphic as functors* **G-Sets** $\to$ **Sets**.

Proof. We define a natural transformation $\alpha\colon U \to \mathrm{Hom}_{\textbf{G-Sets}}(G,-)$ with bijective components α_Ω. For any G-set Ω let α_Ω be the map

$$\begin{aligned} U(\Omega) &\to \mathrm{Hom}_{\textbf{G-Sets}}(G,\Omega) \\ \omega &\mapsto f_\omega \end{aligned}$$

where

$$\begin{aligned} f_\omega\colon G &\to \Omega \\ g &\mapsto \omega g \end{aligned}$$

One immediately sees that for $\omega \in \Omega$ the map f_ω is indeed G-equivariant. We show that α_Ω is bijective with inverse

$$\begin{aligned} \alpha'_\Omega\colon \mathrm{Hom}_{\textbf{G-Sets}}(G,\Omega) &\to U(\Omega) \\ \varphi &\mapsto \varphi(1) \end{aligned}$$

For any $\omega \in \Omega$ we have $\alpha'_\Omega(\alpha_\Omega(\omega)) = \alpha'_\Omega(f_\omega) = f_\omega(1) = \omega$. Conversely, for any morphism $\varphi \in \mathrm{Hom}_{\textbf{G-Sets}}(G,\Omega)$ and all $g \in G$ we have

$$\alpha_\Omega(\alpha'_\Omega(\varphi))(g) = \alpha_\Omega(\varphi(1))(g) = f_{\varphi(1)}(g) = \varphi(1)g = \varphi(g).$$

It remains to show that α is a natural transformation, that is, that for all G-sets Ω and Δ and any morphism $\varphi\colon \Omega \to \Delta$ the following diagram commutes:

$$\begin{array}{ccc} U(\Omega) & \xrightarrow{U(\varphi)} & U(\Delta) \\ \downarrow{\scriptstyle \alpha_\Omega} & & \downarrow{\scriptstyle \alpha_\Delta} \\ \mathrm{Hom}_{\textbf{G-Sets}}(G,\Omega) & \xrightarrow{\varphi_*} & \mathrm{Hom}_{\textbf{G-Sets}}(G,\Delta) \end{array}$$

For any $\omega \in \Omega$ and any $g \in G$ we have

$$\varphi_*(\alpha_\Omega(\omega))(g) = \varphi(f_\omega(g)) = \varphi(\omega g) = \varphi(\omega)g = f_{\varphi(\omega)}(g) = \alpha_\Delta(\varphi(\omega))(g) = \alpha_\Delta(U(\varphi)(\omega))(g).$$

This finishes the proof. □

Definition 1.1.6 (Sub-G-sets)**.** Let Ω be a G-set and let Δ be a subset of Ω. We call Δ a *sub-G-set* of Ω if it is closed under the right action of G, that is, for all $\delta \in \Delta$ we have $\delta g \in \Delta$. If Δ is a sub-G-set of Ω, it is a G-set itself by restricting the action $\Omega \times G \to \Omega$ to $\Delta \times G \to \Delta$, and the set-theoretic inclusion is obviously G-equivariant.

Remark 1.1.7. Note that any sub-G-set Δ of Ω can be written as a union of orbits in Ω: Assume that Δ has a non-empty intersection with some orbit. Then it already contains the whole orbit because it is closed under the right action of G. Conversely, any union of orbits is closed under the right action of G by definition and thus is a sub-G-set.

We now characterize mono- and epimorphisms in **G-Sets**.

Proposition 1.1.8 (Monomorphisms)**.** *A morphism φ in* **G-Sets** *is a monomorphism if and only if it is injective.*

Proof. Let $\varphi\colon \Omega \to \Delta$ be an injective morphism in **G-Sets**. Let $\psi_1, \psi_2\colon \Lambda \to \Omega$ be morphisms in **G-Sets** with $\varphi \circ \psi_1 = \varphi \circ \psi_2$. Since φ is injective, it is a monomorphism in **Sets** and this implies $\psi_1 = \psi_2$. Thus, φ is a monomorphism in **G-Sets**.

Conversely, let $\varphi\colon \Omega \to \Delta$ be a monomorphism in **G-Sets**. Let $\omega_1, \omega_2 \in \Omega$ with $\varphi(\omega_1) = \varphi(\omega_2)$. We want to show that $\omega_1 = \omega_2$. For $t \in \{1,2\}$ we define the morphism

$$\begin{aligned} \psi_t\colon G &\to \Omega \\ g &\mapsto \omega_t g \end{aligned}$$

One immediately sees that ψ_1 and ψ_2 are G-equivariant. We have

$$(\varphi \circ \psi_1)(g) = \varphi(\omega_1 g) = \varphi(\omega_1)g = \varphi(\omega_2)g = \varphi(\omega_2 g) = (\varphi \circ \psi_2)(g)$$

for any $g \in G$, that is, $\varphi \circ \psi_1 = \varphi \circ \psi_2$. Since φ is a monomorphism, we get $\psi_1 = \psi_2$. In particular, we have $\omega_1 = \psi_1(1) = \psi_2(1) = \omega_2$. Thus, φ is injective. □

Proposition 1.1.9 (Epimorphisms)**.** *A morphism φ in* **G-Sets** *is an epimorphism if and only if it is surjective.*

Proof. Let $\varphi\colon \Omega \to \Delta$ be a surjective morphism in **G-Sets**. Let $\psi_1, \psi_2\colon \Delta \to \Lambda$ be morphisms in **G-Sets** with $\psi_1 \circ \varphi = \psi_2 \circ \varphi$. Since φ is surjective, it is an epimorphism in **Sets** and this implies $\psi_1 = \psi_2$. Thus, φ is an epimorphism in **G-Sets**.

Conversely, let $\varphi\colon \Omega \to \Delta$ be an epimorphism in **G-Sets**. Assume that φ is not surjective, that is, we can find $\delta' \in \Delta$ such that there exists no $\omega \in \Omega$ with $\varphi(\omega) = \delta'$. Set $\Lambda := (G\backslash G) \sqcup (G\backslash G)$ and let $x, y \in \Lambda$ with $x \neq y$ be the two elements of Λ. We define two maps ψ_1 and ψ_2:

$$\begin{aligned} \psi_1\colon \Delta &\to \Lambda \\ \delta &\mapsto x \end{aligned} \qquad\qquad \begin{aligned} \psi_2\colon \Delta &\to \Lambda \\ \delta &\mapsto y \quad \text{if } \delta \text{ lies in the orbit of } \delta' \\ \delta &\mapsto x \quad \text{else} \end{aligned}$$

One can quickly check that ψ_1 and ψ_2 are G-equivariant. We claim that $\psi_1 \circ \varphi = \psi_2 \circ \varphi$. For this, it suffices to show that for any $\omega \in \Omega$ its image $\varphi(\omega)$ is not an element of the orbit of δ'. Assume the contrary, that is, there exist $\omega \in \Omega$ and $g \in G$ such that $\varphi(\omega) = \delta' g \in \delta' G$. Then we have

$$\varphi(\omega g^{-1}) = \varphi(\omega) g^{-1} = \delta' g g^{-1} = \delta'$$

which is a contradiction to the choice of δ'. Therefore, we have $\psi_1 \circ \varphi = \psi_2 \circ \varphi$. Since φ is an epimorphism this implies $\psi_1 = \psi_2$. However, this is a contradiction to $\psi_1(\delta') = x \neq y = \psi_2(\delta')$. □

The universal morphisms of equalizers and coequalizers can be obtained as lifts and colifts along mono- and epimorphisms respectively, see Algorithm 7 and Algorithm 9. Thus, we now explicitly construct such lifts and colifts.

Remark 1.1.10 (Lifts along monomorphisms)**.** Let $\varphi\colon \Omega \to \Delta$ be a morphism of G-sets and let $\iota\colon \Lambda \to \Delta$ be a monomorphism of G-sets with $\varphi(\Omega) \subseteq \iota(\Lambda)$. Then ι is injective and thus we can find a set-theoretic lift of φ along ι, that is, a map $\psi\colon \Omega \to \Lambda$ of sets such that the following diagram commutes:

$$\begin{array}{ccc} \Lambda & \xhookrightarrow{\iota} & \Delta \\ {\scriptstyle\psi}\uparrow & \nearrow_{\varphi} & \\ \Omega & & \end{array}$$

The image of $\omega \in \Omega$ under ψ is the uniquely determined element $\lambda \in \Lambda$ with $\iota(\lambda) = \varphi(\omega)$. We want to show that ψ is G-equivariant. For all $\omega \in \Omega$ and $g \in G$ we have

$$\iota(\psi(\omega g)) = \varphi(\omega g) = \varphi(\omega) g = \iota(\psi(\omega)) g = \iota(\psi(\omega) g).$$

Since ι is a monomorphism, this implies $\psi(\omega g) = \psi(\omega) g$, and this finishes the proof. Often, Λ will be a sub-G-set of Δ and ι the inclusion.

Remark 1.1.11 (Colifts along epimorphisms)**.** Let $\varphi\colon \Omega \to \Delta$ be a morphism of G-sets and let $\pi\colon \Omega \to \Lambda$ be an epimorphism of G-sets such that any two elements of Ω have the same image under φ whenever they have the same image under π. Then π is surjective and thus we can find a set-theoretic colift of φ along π, that is, a map $\psi\colon \Lambda \to \Delta$ of sets such that the following diagram commutes:

$$\begin{array}{ccc} \Omega & \xrightarrow{\pi} & \Lambda \\ & {\scriptstyle\varphi}\searrow & \downarrow{\scriptstyle\psi} \\ & & \Delta \end{array}$$

To obtain the image of $\lambda \in \Lambda$ under ψ, choose $\omega \in \Omega$ with $\pi(\omega) = \lambda$ and set $\psi(\lambda) := \varphi(\omega)$. By assumption this assignment is independent of the choice of ω. We show that ψ is G-equivariant: For $\lambda \in \Lambda$ and $g \in G$ we can choose $\omega \in \pi^{-1}(\lambda) \neq \varnothing$ and get

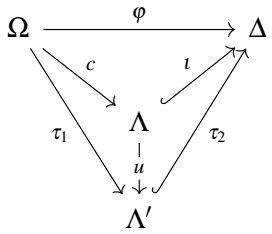
$$\psi(\lambda) g = \psi(\pi(\omega)) g = \varphi(\omega) g = \varphi(\omega g) = \psi(\pi(\omega g)) = \psi(\pi(\omega) g) = \psi(\lambda g).$$

This finishes the proof.

We will need the concept of images for finding *connected components* in Algorithm 9. Thus, we now show that in **G-Sets** all images exist.

Proposition 1.1.12 (Images)**.** *In* **G-Sets** *all images exist.*

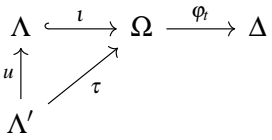
Proof. Let $\varphi\colon \Omega \to \Delta$ be a morphism of G-sets and let $\tau_1\colon \Omega \to \Lambda'$ and $\tau_2\colon \Lambda' \to \Delta$ be morphisms of G-sets such that τ_2 is a monomorphism and $\tau_2 \circ \tau_1 = \varphi$. Let Λ be the canonical set-theoretic image of φ, $c\colon \Omega \to \Lambda$ the set-theoretic coastriction, $\iota\colon \Lambda \hookrightarrow \Delta$ the set-theoretic embedding and $u\colon \Lambda \to \Lambda'$ the unique set-theoretic morphism such that $u \circ c = \tau_1$ and $\tau_2 \circ u = i$. That is, we have the following commutative diagram in **Sets**:



We have to show that this is a commutative diagram in **G-Sets**. Let $y \in \Lambda$. Then there exists $x \in \Omega$ such that $\varphi(x) = y$. Thus, we have $yg = \varphi(x)g = \varphi(xg) \in \Lambda$ for all $g \in G$. Hence, Λ is a sub-G-set of Ω and therefore ι is G-equivariant. The coastriction c is the lift of φ along the monomorphism ι and the universal morphism u is the lift of ι along the monomorphism τ_2. Therefore, both are G-equivariant. □

Proposition 1.1.13 (Equalizers)**.** *In* **G-Sets** *all equalizers exist.*

Proof. Let $\varphi_t\colon \Omega \to \Delta$ be morphisms of G-sets for $t \in \{1,\dots,s\}$ and let $\tau\colon \Lambda' \to \Omega$ be a morphism of G-sets with $\varphi_a \circ \tau = \varphi_b \circ \tau$ for all $a,b \in \{1,\dots,s\}$. Let $\Lambda \subseteq \Omega$ be the canonical set-theoretic equalizer, let $\iota\colon \Lambda \to \Omega$ be the set-theoretic inclusion and let $u\colon \Lambda' \to \Lambda$ be the unique set-theoretic morphism such that $\iota \circ u = \tau$. That is, we have the following diagram in **Sets**:



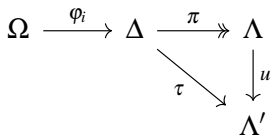
We have to show that this is a diagram in **G-Sets**. For all $\lambda \in \Lambda$, $g \in G$ and $a,b \in \{1,\dots,s\}$ we have

$$\varphi_a(\lambda g) = \varphi_a(\lambda)g = \varphi_b(\lambda)g = \varphi_b(\lambda g).$$

This implies that $\lambda g \in \Lambda$ for all $\lambda \in \Lambda$ and $g \in G$, that is, Λ is a sub-G-set of Ω. Therefore, the inclusion ι is G-equivariant. Moreover, u is G-equivariant since it is the lift of τ along the monomorphism ι. □

Proposition 1.1.14 (Coequalizers)**.** *In* **G-Sets** *all coequalizers exist.*

Proof. Let $\varphi_t\colon \Omega \to \Delta$ be morphisms of G-sets for $t \in \{1,\dots,s\}$ and let $\tau\colon \Delta \to \Lambda'$ be a morphism of G-sets with $\tau \circ \varphi_a = \tau \circ \varphi_b$ for all $a,b \in \{1,\dots,s\}$. Let Λ be the the canonical set-theoretic coequalizer, let $\pi\colon \Delta \to \Lambda$ be the set-theoretic projection and let $u\colon \Lambda \to \Lambda'$ be the unique set-theoretic morphism such that $u \circ \pi = \tau$. That is, we have the following diagram in **Sets**:



We have to show that this is a diagram in **G-Sets**. The coequalizer Λ is the set of equivalence classes of the transitive hull of the relation $\sim$ on Δ defined as follows: for $\delta_1, \delta_2 \in \Delta$ we set $\delta_1 \sim \delta_2$ if and

only of there exists $\omega \in \Omega$ and $a,b \in \{1,\dots,s\}$ such that $\varphi_a(\omega) = \delta_1$ and $\varphi_b(\omega) = \delta_2$. This relation is compatible with the action of G: for $\delta_1 \sim \delta_2$, $\omega \in \Omega$ and $a,b \in \{1,\dots,s\}$ as before and any $g \in G$ we have $\varphi_a(\omega g) = \delta_1 g$ and $\varphi_b(\omega g) = \delta_2 g$, and thus $\delta_1 g \sim \delta_2 g$. Therefore, we can define an action of G on Λ by acting on representatives. Since π maps any element of Δ to its equivalence class in Λ, we immediately get that π is G-equivariant. Moreover, u is G-equivariant since it is the colift of τ along the epimorphism π. □

We will now give a definition of the *table of marks*, which we will need for the third algorithm in the final section.

Definition 1.1.15 (Mark)**.** Let Ω be a G-set. The *mark* of Ω is the map

$$\begin{aligned} \beta_\Omega \colon \mathscr{U} &\to \mathbb{Z}_{\geq 0} \\ U_i &\mapsto |\mathrm{fix}_{U_i}(\Omega)|. \end{aligned}$$

As shorthand notations, we set $\beta_\Omega(i) := \beta_\Omega(U_i)$, and $\beta_V := \beta_{V\backslash G}$ for a subgroup $V \leq G$.

Definition 1.1.16 (Table of marks)**.** The matrix

$$(\beta_{U_i}(j))_{1\leq i,j\leq k} = \left(|\mathrm{fix}_{U_j}(U_i\backslash G)|\right)_{1\leq i,j\leq k}$$

is called *table of marks* of G.

Remark 1.1.17 (Properties of marks)**.** Two G-sets are isomorphic if and only they have have the same mark. The table of marks is a lower triangular matrix with non-zero diagonal entries. Let Ω_1, Ω_2, and $\Omega = \bigsqcup_{i=1}^{k} \bigsqcup_{\ell=1}^{m_i} U_i\backslash G$ be G-sets. Then it is easy to see that $\beta_{\Omega_1\times\Omega_2} = \beta_{\Omega_1} \cdot \beta_{\Omega_2}$ and $\beta_{\Omega_1\sqcup\Omega_2} = \beta_{\Omega_1} + \beta_{\Omega_2}$ with pointwise multiplication and addition. In particular, $\beta_\Omega = \sum_{i\in\mathscr{I}} m_i\beta_{U_i}$. Additionally, the marks β_{U_i} with $i \in \mathscr{I}$ are linearly independent over $\mathbb{Z}$. Thus, we can find the multiplicities m_i by decomposing the mark β_Ω with regard to the marks β_{U_i}.

1.1.2 The category Skeletal-G-Sets

We now want to define the category **Skeletal-G-Sets** in a way suitable for implementation in CAP and thus have to find representations of objects in **G-Sets** up to isomorphism and their morphisms. For this, first recall the following proposition:

Proposition 1.1.18 (Classification of finite G-sets)**.** *Let Ω be a finite G-set and $\omega_1,\dots,\omega_s \in \Omega$ a system of representatives of the orbits in Ω. Then $\Omega \cong \Delta := \bigsqcup_{t=1}^{s} \mathrm{Stab}_G(\omega_i)\backslash G$ as G-sets, where G acts on Δ by right multiplication. Moreover, we can choose the ω_i in such a way that $\mathrm{Stab}_G(\omega_i) \in \mathscr{U}$. Thus, $\Omega \cong \bigsqcup_{t=1}^{s} U_{i_t}\backslash G$ for suitable $i_t \in \mathscr{I}$. Moreover, the indices i_t are uniquely determined up to permutation.*

Remark 1.1.19 (Representation of G-sets up to isomorphism)**.** By Proposition 1.1.18, any G-set Ω is isomorphic to a disjoint union Δ of right cosets $U_i\backslash G$ with $U_i \in \mathscr{U}$, where each $U_i\backslash G$ occurs with a uniquely determined multiplicity $m_i \in \mathbb{Z}_{\geq 0}$. That is, we have

$$\Omega \cong \Delta = \bigsqcup_{i=1}^{k} m_i(U_i\backslash G) := \bigsqcup_{i=1}^{k} \bigsqcup_{\ell=1}^{m_i} (U_i\backslash G)_\ell$$

with $(U_i\backslash G)_\ell := U_i\backslash G$ for all $i \in \mathscr{I}$. Thus, a possible skeletal representation of both Ω and Δ is given by the k-tuple $(m_1,\dots,m_k)$. In this setting, we set

$$(1_i)_\ell := U_i 1 \in (U_i\backslash G)_\ell \subseteq \Delta \text{ and}$$
$$(U_i g)_\ell := U_i g \in (U_i\backslash G)_\ell \subseteq \Delta.$$

In [Mic18], we see that taking the disjoint union with the canonical embeddings gives a coproduct in **G-Sets**. Thus, the set $(U_i\backslash G)_\ell$ is a transitive cofactor of Δ and we call the pair (i,ℓ) its *position*.

Remark 1.1.20 (Representation of morphisms of G-sets)**.** Let $\Omega = \bigsqcup_{i=1}^{k}\bigsqcup_{\ell=1}^{m_i}(U_i\backslash G)_\ell$ and $\Delta = \bigsqcup_{j=1}^{k}\bigsqcup_{r=1}^{n_j}(U_j\backslash G)_r$ be G-sets. Any morphism $\varphi\colon \Omega \to \Delta$ of G-sets is uniquely determined by the images of the elements $(1_i)_\ell \in \Omega$: For any $U_i g \in (U_i\backslash G)_\ell$ we have

$$\varphi(U_i g) = \varphi(U_i 1)g = \varphi((1_i)_\ell)g$$

since φ is G-equivariant. Conversely, if we choose $U_{j_{(i,\ell)}}g_{(i,\ell)} \in \Delta$ for every position (i,ℓ) of a transitive cofactor of Ω and set

$$\varphi((1_i)_\ell) := U_{j_{(i,\ell)}}g_{(i,\ell)},$$

we can extend this to a morphism of G-sets by setting

$$\varphi(U_i g) := \varphi((1_i)_\ell)g \text{ for } U_i g \in (U_i\backslash G)_\ell \subseteq \Omega$$

as long as this extension is well-defined. The extension is well-defined if and only if it is independent of the choice of the representative of $U_i g$, that is, if for all $ug \in U_i g$ we have

$$U_{j_{(i,\ell)}}g_{(i,\ell)}ug = \varphi((1_i)_\ell)ug = \varphi((1_i)_\ell)g = U_{j_{(i,\ell)}}g_{(i,\ell)}g.$$

This in turn is equivalent to

$$U_{j_{(i,\ell)}}g_{(i,\ell)}ug_{(i,\ell)}^{-1} = U_{j_{(i,\ell)}}$$

for all $u \in U_i$, that is, we have

$$g_{(i,\ell)}ug_{(i,\ell)}^{-1} \in U_{j_{(i,\ell)}}$$

for all $u \in U_i$. Thus, the condition for the images $U_{j_{(i,\ell)}}g_{(i,\ell)}$ is that we have

$$g_{(i,\ell)}U_i g_{(i,\ell)}^{-1} \subseteq U_{j_{(i,\ell)}}.$$

Note that taking a different representative of $U_{j_{(i,\ell)}}g_{(i,\ell)}$ does not change this condition.

Therefore, we can represent a morphisms $\varphi\colon \Omega \to \Delta$ as a k-tuple L with the following structure: The i-th entry of L is an m_i-tuple L_i. The ℓ-th entry of L_i is a triple $(r,U_j g,j)$ with integers $1 \le j \le k$ and $1 \le r \le n_j$ and $g \in G$ such that $gU_i g^{-1} \subseteq U_j$, which we interpret as

$$\varphi((1_i)_\ell) := (U_j g)_r \in (U_j\backslash G)_r \subseteq \Delta.$$

We call $(r,U_j g,j)$ the *component of* φ *at position* (i,ℓ) and the pair (j,r) the *target position* of the component. Note that the implementation in CAP also accepts the triple (r,g,j) instead of $(r,U_j g,j)$ for simpler notation. For an example, see Example 1.1.25. However, we first turn this into a precise definition.

Definition 1.1.21 (The category **Skeletal-G-Sets**)**.** We now define the category **Skeletal-G-Sets**: Its objects are k-tuples of non-negative integers. A morphism between $\Omega = (m_1, \ldots, m_k)$ and $\Delta = (n_1, \ldots, n_k)$ is given by a k-tuple L with the following structure: The i-th entry of L is an m_i-tuple L_i. The ℓ-th entry of L_i is a triple $(r, U_j g, j)$, where j and r are integers with $1 \leq j \leq k$ and $1 \leq r \leq n_j$ and where $g \in G$ such that $gU_i g^{-1} \subseteq U_j$. The remarks above show that this indeed represents the skeleton of **G-Sets**.

Remark 1.1.22 (Notation)**.** For better readability we will always use square brackets for the tuples defining objects and morphisms of **Skeletal-G-Sets**, with the only exception that for the triples in the definition of a morphism we will use round brackets as usual. In particular, [] is always the 0-tuple whereas () is the identity element of some S_n used in examples.

Definition 1.1.23. We define the *realization functor* R: **Skeletal-G-Sets** $\to$ **G-Sets** which assigns to a k-tuple in **Skeletal-G-Sets** its actual G-set and to a morphism the actual morphism of G-sets using the interpretations in the remarks above. Conversely, we define the *abstraction functor* A: **G-Sets** $\to$ **Skeletal-G-Sets** which assigns to a G-set respectively a morphism its representing k-tuple as in the remarks above. However, for ease of notation we do not distinguish between an object Ω in **Skeletal-G-Sets**, its realization as a G-set $R(\Omega)$ and the corresponding underlying set $U(R(\Omega))$ if it is not relevant in the given context. In particular, we write $\omega \in \Omega$ instead of $\omega \in U(R(\Omega))$ and $\varphi(\omega)$ instead of $U(R(\varphi))(\omega)$. Note that R and A are the canonical equivalences between **G-Sets** and its skeleton.

Definition 1.1.24 (Positions of an object in **Skeletal-G-Sets**)**.** Let $\Omega = [m_1, \ldots, m_k]$ be an object in **Skeletal-G-Sets**. A *position* of Ω is a position of a transitive cofactor of $R(\Omega)$, that is, an element of the set $\{(i, \ell) \mid 1 \leq i \leq k, 1 \leq \ell \leq m_i\}$.

Example 1.1.25. Let $G = S_3$ and let $\Omega = [2,1,0,0]$ and $\Delta = [1,2,0,0]$ be two objects in **Skeletal-G-Sets**. Let $\varphi\colon \Omega \to \Delta$ be the morphism given by by

$$[[], [(1, U_2(123), 2), (1, U_2(), 2)], [(2, U_2(), 2)], []].$$

We visualize this as follows:

$$\begin{array}{ccc}
R(\Omega) & & R(\Delta) \\
= & & = \\
\{1\}\backslash S_3 & & \{1\}\backslash S_3 \\
 & \overset{(123)}{\searrow} & \\
\sqcup & & \sqcup \\
\{1\}\backslash S_3 & \xrightarrow{()} & U_2\backslash S_3 \\
\sqcup & & \sqcup \\
U_2\backslash S_3 & \xrightarrow{()} & U_2\backslash S_3
\end{array}$$

The G-sets $R(\Omega)$ and $R(\Delta)$ are the disjoint unions of the sets in the first and second column respectively. The components $(1,U_2(123),2)$, $(1,U_2(),2)$, and $(2,U_2(),2)$ of φ correspond exactly to the arrows. The positions of Ω are $(1,1)$, $(1,2)$ and $(2,1)$. The positions of Δ are $(1,1)$, $(2,1)$ and $(2,2)$.

Remark 1.1.26. Recall that two G-sets are isomorphic if and only if they have the same mark. Thus, we could have used the marks as objects of **Skeletal-G-Sets**. However, for defining morphisms and visualizing the corresponding G-sets it is easier to use the "coordinates" of the marks with regard to the marks β_{U_i}, that is, the multiplicities m_i as in Remark 1.1.17.

1.1.3 Algorithms in Skeletal-G-Sets

We now want to explain some of the algorithms of our implementation of **Skeletal-G-Sets**. In particular, we are interested in algorithms for finite equalizers and coequalizers. We start with some basic algorithms which are explained in more detail in [Mic18].

Remark 1.1.27 (Composition of morphisms in **Skeletal-G-Sets**)**.** Let $\varphi\colon \Omega \to \Delta$ and $\varphi'\colon \Delta \to \Lambda$ be morphisms in **Skeletal-G-Sets** with $\Omega = [m_1,\dots,m_k]$, $\Delta = [n_1,\dots,n_k]$ and $\Lambda = [o_1,\dots,o_k]$. We want to determine $\psi := \varphi' \circ \varphi$. For this, we have to compute the images $\varphi'(\varphi((1_i)_\ell))$ for all positions (i,ℓ) of Ω. Fix a position (i,ℓ) of Ω. Let $(r_1,U_{j_1}g_1,j_1)$ be the component of φ at position (i,ℓ) and let $(r_2,U_{j_2}g_2,j_2)$ be the component of φ' at position (j_1,r_1). Then, by definition we have

$$\varphi'(\varphi((1_i)_\ell)) = \varphi'((U_{j_1}g_1)_{r_1}) = \varphi'((U_{j_1}1)_{r_1})g_1 = \varphi'((1_{j_1})_{r_1})g_1 = (U_{j_2}g_2)_{r_2}g_1 = (U_{j_2}g_2g_1)_{r_2},$$

which gives the component $(r_2,U_{j_2}g_2g_1,j_2)$. In particular, to compose arrows in a diagram like the one above we only have to multiply the labels (in reverse order) using the multiplication in G.

Remark 1.1.28 (Equality of morphisms in **Skeletal-G-Sets**)**.** Two morphisms $\varphi\colon \Omega \to \Delta$ and $\varphi'\colon \Omega \to \Delta$ in **Skeletal-G-Sets** are equal if and only if $R(\varphi) = R(\varphi')$ as maps. We want to find an easy way to check this condition. For this, observe that $R(\varphi) = R(\varphi')$ if and only if $R(\varphi)|_C = R(\varphi')|_C$ for all transitive cofactors C of $R(\Omega)$. Let C be such a cofactor and (i,ℓ) its position. If $R(\varphi)|_C = R(\varphi')|_C$, then in particular we have $R(\varphi)((1_i)_\ell) = R(\varphi')((1_i)_\ell)$. Conversely, if $R(\varphi)((1_i)_\ell) = R(\varphi')((1_i)_\ell)$ we get $R(\varphi)((U_ig)_\ell) = R(\varphi')((U_ig)_\ell)$ for all g in G since $R(\varphi)$ and $R(\varphi')$ are G-equivariant. Since C is transitive, this implies $R(\varphi)|_C = R(\varphi')|_C$.

Summing up, we get that φ and φ' are equal if and only if for all positions p of Ω the components of φ and φ' at position p are the same. Note that we might still have different representations if we consider representatives of cosets instead of the actual cosets.

Now we explain the algorithms which we need as a preparation for the algorithms computing equalizers and coequalizers. The method names of the implementations in CAP are set in a `monospace font`.

Remark 1.1.29 (Representation of sub-G-sets in **Skeletal-G-Sets**)**.** Let $\Omega = [m_1,\dots,m_k]$ be an object in **Skeletal-G-Sets** and Λ a sub-G-set of $R(\Omega)$. We know that Λ is a union of orbits in $R(\Omega)$ and the orbits in $R(\Omega)$ are exactly its transitive cofactors. Thus, Λ is a union of transitive cofactors of $R(\Omega)$ and can be represented as the set P of the positions of these cofactors. Additionally, we can immediately read of its representation as an object in **Skeletal-G-Sets**: for each $i \in \mathscr{I}$ we have to count how many transitive cofactors of Λ are equal to $U_i\backslash G$, that is, how many elements in P have first component i. Finally, we determine the representation of the embedding $\iota\colon \Lambda \hookrightarrow R(\Omega)$. For this, sort the elements of P in lexicographical order. Let (i,ℓ) be a position of Λ and let (i,r) be the ℓ-th element in P which has first component i. Then the component of ι at position (i,ℓ) is given by (r,U_i1,i).

We make this precise in the next definition and in Algorithm 1.

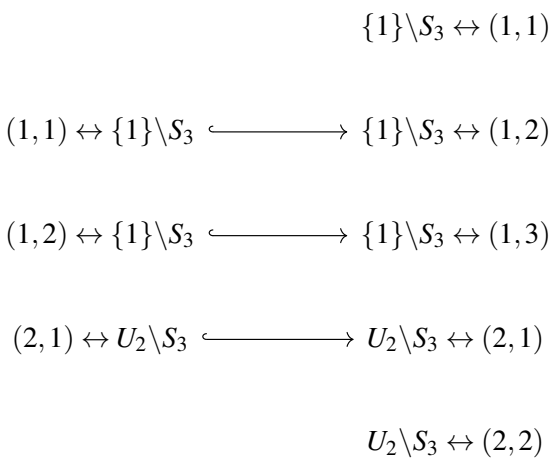


Figure 1.1: A visualization of Example 1.1.31

Definition 1.1.30 (Sub-G-sets of objects in **Skeletal-G-Sets**)**.** Let $\Omega = [m_1, \ldots, m_k]$ be an object in **Skeletal-G-Sets**. A *sub-G-set* of Ω is a subset of the set of positions of Ω, that is, a subset of $\{(i,\ell) \mid 1 \leq i \leq k, 1 \leq \ell \leq m_i\}$. Using the translation in Remark 1.1.29 we see that the sub-G-sets of Ω correspond exactly to the sub-G-sets of $R(\Omega)$. Let P be a sub-G-set of Ω, let Λ be the corresponding sub-G-set of $R(\Omega)$ and let $\iota\colon \Lambda \hookrightarrow R(\Omega)$ be the embedding of Λ. Then we call $A(\iota)$ the *embedding* of P. Let $\varphi\colon \Omega \to \Delta$ be a morphism in **Skeletal-G-Sets**. Then we call $\varphi \circ A(\iota)$ the *restriction* of φ to P.

Example 1.1.31 (Sub-G-sets in **Skeletal-G-Sets**)**.** Let $G = S_3$, $\Omega = [3,2,0,0]$ an object in **Skeletal-G-Sets** and $P = \{(1,2),(1,3),(2,1)\}$ a sub-G-set of Ω. We visualize this in Figure 1.1. The representation of P as an object in **Skeletal-G-Sets** is $[2,1,0,0]$, and the components of the embedding ι of P are $(2,(),1)$, $(3,(),1)$ and $(1,(),2)$.

Algorithm 1: `EmbeddingOfPositions`

Input: a sub-G-set P of an object T in **Skeletal-G-Sets**
Output: the embedding of P

foreach $i = 1, \ldots, k$ **do**
  let L_i be the set of positions in P with first component i;
  sort the elements of L_i by their second component;
the source S of the embedding is given by the tuple $(|L_i|)_i$;
let ι be the morphism $S \to T$ with components defined as follows:
foreach *position* $p = (i,\ell)$ *of* S **do**
  the component of ι at position p is $(\ell', U_i 1, i)$ where ℓ' is the second component of the ℓ-th entry of L_i;
return ι;

Remark 1.1.32 (Preimages in **Skeletal-G-Sets**)**.** Let Ω and Δ be objects in **Skeletal-G-Sets** and let $\varphi\colon \Omega \to \Delta$ be a morphism in **Skeletal-G-Sets**. Let P' be a sub-G-set of Δ and let Λ' be the

corresponding sub-G-set of $R(\Delta)$. We want to describe the set-theoretic preimage Λ of Λ' under $R(\varphi)$. For any $(U_i g)_\ell \in \Omega$ we have

$$\begin{aligned} & (U_i g)_\ell \in \Lambda \\ \Leftrightarrow \quad & \varphi((U_i g)_\ell) \in \Lambda' \\ \Leftrightarrow \quad & \varphi((U_i 1)_\ell) g \in \Lambda' \\ \Leftrightarrow \quad & \varphi((1_i)_\ell) \in \Lambda' \end{aligned}$$

and this holds true if and only if the target position of the component of φ at position (i,ℓ) is an element of P'. Thus, Λ is a sub-G-set of $R(\Omega)$ and corresponds to the sub-G-set P of Ω with the following property: a position p of Ω is in P if and only if the target position of the component of φ at position p is an element of P'. We make this precise in Algorithm 2.

Algorithm 2: `PreimagePositions`

Input: a morphism $\varphi\colon S \to T$ and a sub-G-set P' of T corresponding to a sub-G-set Λ' of $R(T)$
Output: the sub-G-set P of S corresponding to the preimage of Λ' under $R(\varphi)$

```
1 let P be the empty set;
2 foreach position p_S of S do
3     if the target position of the component of φ at position p_S is an element of P' then
4         add p_S to P;
5 return P;
```

Remark 1.1.33 (Lifts along monomorphisms in **Skeletal-G-Sets**)**.** Let $\varphi\colon \Omega \to \Delta$ be a morphism in **Skeletal-G-Sets** and let $\iota\colon \Lambda \hookrightarrow \Delta$ be a monomorphism in **Skeletal-G-Sets** with $R(\varphi)(\Omega) \subseteq R(\iota)(\Lambda)$. By Remark 1.1.10, the lift $\psi\colon R(\Omega) \to R(\Lambda)$ of $R(\varphi)$ along $R(\iota)$ exists, and makes the following diagram commute:

$$\begin{array}{ccc} R(\Lambda) & \xhookrightarrow{R(\iota)} & R(\Delta) \\ {\scriptstyle\psi}\uparrow & \nearrow_{R(\varphi)} & \\ R(\Omega) & & \end{array}$$

We want to find the representation of ψ in **Skeletal-G-Sets**. For this, we have to compute $\psi((1_i)_\ell)$ for all positions (i,ℓ) of Ω. By construction, $\psi((1_i)_\ell)$ is the unique preimage of $(U_j g)_r := \varphi((1_i)_\ell)$ under $R(\iota)$. Let (s,t) be the position of the component of ι with target position (j,r), that is, a position of Λ such that the component of ι at position (s,t) is $(r, U_j h, j)$ for some $h \in G$. Then we have

$$\iota((U_s h^{-1} g)_t) = (U_j h)_r h^{-1} g = (U_j g)_r = \varphi((1_i)_\ell).$$

Thus, we get $\psi((1_i)_\ell) = (U_s h^{-1} g)_t$ which is represented by the triple $(t, U_s h^{-1} g, s)$. We make this precise in Algorithm 3.

Algorithm 3: `LiftAlongMonomorphism`

Input: a monomorphism ι and a morphism φ in **Skeletal-G-Sets** such that a lift of φ along ι exists

Output: such a lift ψ

let S be the source of φ;
let T be the source of ι;
let ψ be the morphism $S \to T$ with components defined as follows:
foreach *position* $p = (i,\ell)$ *of* S **do**
    let $(r, U_j g, j)$ be the component of φ at position p;
    let (s,t) be the (unique) preimage position of $\{(j,r)\}$ under ι;
    let $h \in G$ such that $(r, U_j h, j)$ is the component of ι at position (s,t);
    the component of ψ at position p is $(t, U_s h^{-1} g, s)$;
return ψ;

Remark 1.1.34 (Colifts along epimorphisms in **Skeletal-G-Sets**). Let $\varphi\colon \Omega \to \Delta$ be a morphism in **Skeletal-G-Sets** and let $\pi\colon \Omega \twoheadrightarrow \Lambda$ be an epimorphism in **Skeletal-G-Sets** such that any two elements of $R(\Omega)$ have the same image under $R(\varphi)$ whenever they have the same image under $R(\pi)$. By Remark 1.1.11, the colift $\psi\colon R(\Lambda) \to R(\Delta)$ of $R(\varphi)$ along $R(\pi)$ exists, and makes the following diagram commute:

$$\begin{array}{ccc} R(\Omega) & \xrightarrow{R(\pi)} & R(\Lambda) \\ & \searrow^{R(\varphi)} & \downarrow{\scriptstyle \psi} \\ & & R(\Delta) \end{array}$$

We want to find the representation of ψ in **Skeletal-G-Sets**. For this, we have to compute $\psi((1_i)_\ell)$ for all positions (i,ℓ) of Λ. By construction, $\psi((1_i)_\ell)$ is the image under $R(\varphi)$ of the preimage of $(1_i)_\ell$ under $R(\pi)$. Let (s,t) be a preimage position of $\{(i,\ell)\}$ under π, that is, a position of Ω such that the target position of the component of π at position (s,t) is (i,ℓ). Let $(\ell, U_i h, i)$ be the component of π at position (s,t). Then we have

$$\pi((U_s h^{-1})_t) = (U_i h)_\ell h^{-1} = (1_i)_\ell,$$

so $(U_s h^{-1})_t$ is a preimage of $(1_i)_\ell$ under $R(\pi)$. Now let $(r, U_j g, j)$ be the component of φ at position (s,t). Then we have

$$\varphi((U_s h^{-1})_t) = (U_j g)_r h^{-1} = (U_j g h^{-1})_r$$

which is represented by the triple $(r, U_j g h^{-1}, j)$. We make this precise in Algorithm 4.

Remark 1.1.35 (Images in **Skeletal-G-Sets**). By Proposition 1.1.12, all images in G-set exist. Therefore, also in **Skeletal-G-Sets** all images exist. Let Ω and Δ be objects in **Skeletal-G-Sets** and let $\varphi\colon \Omega \to \Delta$ be a morphism. We want to find an image Λ of φ and its embedding $\iota\colon \Lambda \hookrightarrow \Delta$. By Proposition 1.1.12, we know that $R(\Lambda)$ is a sub-G-set of $R(\Delta)$, that is, a disjoint union of transitive cofactors of $R(\Delta)$. We observe the following fact: Let C be a transitive cofactor of $R(\Delta)$. If there exists $(1_i)_\ell \in \Omega$ such that $\varphi((1_i)_\ell) \in C$, then by the G-equivariance of $R(\varphi)$ and since C is a transitive G-set, we already have that C is a subset of the set-theoretic image. Conversely, if C is a subset of the set-theoretic image, there exists an $\omega \in \Omega$ such that $\varphi(\omega) \in C$. Write $\omega = (1_i)_\ell g$ for some $(1_i)_\ell \in \Omega$.

Algorithm 4: `ColiftAlongEpimorphism`

Input: an epimorphism π and a morphism φ in **Skeletal-G-Sets** such that a colift of φ along π exists
Output: such a lift ψ

let S be the range of π;
let T be the range of φ;
let ψ be the morphism $S \to T$ with components defined as follows:
foreach *position* (i,ℓ) *of* S **do**
    let (s,t) be a preimage position of $\{(i,\ell)\}$ under π;
    let $h \in G$ such that $(\ell, U_i h, i)$ is the component of π at position (s,t);
    let $(r, U_j g, j)$ be the component of φ at position (s,t);
    the component of ψ at position p is $(r, U_j g h^{-1}, j)$;
return ψ;

Then $\varphi((1_i)_\ell) = \varphi(\omega g^{-1}) = \varphi(\omega) g^{-1} \in C$ since C is a transitive G-set. Thus, $R(\Lambda)$ is the sub-G-set of $R(\Delta)$ corresponding to the sub-G-set P of Δ which consists of the target positions of the components of φ, and $\iota : \Lambda \hookrightarrow \Delta$ is the embedding of P. We make this precise in Algorithm 5 and Algorithm 6.

Algorithm 5: `ImagePositions`

Input: a morphism $\varphi\colon S \to T$
Output: the sub-G-set P of T corresponding to the sub-G-set $\mathrm{im}(R(\varphi))$ of $R(T)$

let P be the empty set;
foreach *position* p_T *of* T **do**
    if *there exists a position* p_S *of* S *such that the component of* φ *at position* p_S *has target position* p_T **then**
        add p_T to P;
return P;

Remark 1.1.36 (Mono- and epimorphisms in **Skeletal-G-Sets**). Let $\varphi\colon \Omega \to \Delta$ be a morphism in **Skeletal-G-Sets**. We want to find algorithms to determine if φ is a monomorphism respectively an epimorphism.

If φ is a monomorphism, then $R(\varphi)$ is injective and thus bijective onto its image. Therefore, we have $\mathrm{im}(\varphi) = \Omega$ because **Skeletal-G-Sets** is a skeletal category. Conversely, if we have $\mathrm{im}(\varphi) = \Omega$, then in particular $|\mathrm{im}(R(\varphi))| = |R(\Omega)|$ as sets and this implies that $R(\varphi)$ is injective, that is, φ is a monomorphism.

Since $U(R(\varphi))$ is a map of sets, it is surjective if and only if $\mathrm{im}(U(R(\varphi))) = U(R(\Delta))$ and this holds true if and only if $\mathrm{im}(\varphi) = \Delta$. Thus, φ is an epimorphism if and only if $\mathrm{im}(\varphi) = \Delta$.

Finally, we are able to explain the algorithms for computing equalizers and coequalizers.

Remark 1.1.37 (Equalizers in **Skeletal-G-Sets**). By Proposition 1.1.13, all equalizers in **G-Sets** exist. Therefore, also in **Skeletal-G-Sets** all equalizers exist. Let Ω and Δ be objects in **Skeletal-G-Sets**

Algorithm 6: Images in **Skeletal-G-Sets**

Input: a morphism $\varphi\colon \Omega \to \Delta$ in **Skeletal-G-Sets** and optionally a test object Λ' and test morphisms $\tau_1\colon \Omega \to \Lambda'$ and $\tau_2\colon \Lambda' \to \Delta$ such that τ_2 is a monomorphism and $\tau_2 \circ \tau_1 = \varphi$

Output: an image Λ of φ with an embedding $\iota\colon \Lambda \to \Delta$ and a coastriction $c\colon \Omega \to \Lambda$; additionally the universal morphism $\psi\colon \Lambda \to \Lambda'$ if a test object is given

`ImageEmbedding` and `ImageObject`:
let ι be the embedding of the sub-G-set `ImagePositions`(φ) of Δ;
let Λ be the source of ι;

`CoastrictionToImage` (automatically derived by CAP):
let c be the lift of φ along the monomorphism ι;

`UniversalMorphismFromImage` (automatically derived by CAP):
if *a test object is given* **then**
  let ψ be the lift of ι along the monomorphism τ_2;

return Λ, ι, and c, and additionally ψ if a test object is given;

and let $\varphi_t\colon \Omega \to \Delta$ be morphisms in **Skeletal-G-Sets** for $t \in \{1,\dots,s\}$. We want to find an equalizer Λ of the φ_i and its embedding ι into Ω. By Proposition 1.1.13, we know that $R(\Lambda)$ is a sub-G-set of $R(\Omega)$, that is, a disjoint union of transitive cofactors of $R(\Omega)$. Concretely, it contains exactly the transitive cofactors C of $R(\Omega)$ such that $R(\varphi_a)|_C = R(\varphi_b)|_C$ for all $a,b \in \{1,\dots,s\}$. We have seen in Remark 1.1.28 that $R(\varphi_a)|_C = R(\varphi_b)|_C$ is equivalent to $\varphi_a((1_i)_\ell) = \varphi_b((1_i)_\ell)$ where (i,ℓ) is the position of C. Therefore, $R(\Lambda)$ is the sub-G-set of $R(\Omega)$ corresponding to the sub-G-set P of Ω with the following property: a position p of Ω is an element of P if and only if the target positions of the φ_t at position p are pairwise equal. Moreover, $\iota\colon \Lambda \hookrightarrow \Omega$ is the embedding of P. We make this precise in Algorithm 7.

Example 1.1.38 (Coequalizers in **Skeletal-G-Sets**)**.** Before we explain our algorithm for computing a coequalizer, we give a motivating example. Let $G = S_3$ and let $\Omega = [4,0,0,0]$ and $\Delta = [0,4,0,0]$ be objects in **Skeletal-G-Sets**. Let $\varphi_1, \varphi_2\colon \Omega \to \Delta$ be morphisms in **Skeletal-G-Sets** represented by

$$[[(1,U_2(),2),(1,U_2(),2),(3,U_2(),2),(3,U_2(),2)],[],[],[]]$$

and

$$[[(2,U_2(123),2),(2,U_2(),2),(4,U_2(),2),(4,U_2(),2)],[],[],[]]$$

respectively. Figure 1.2 shows a visualization. The dotted and dashed arrows correspond to φ_1 and φ_2 respectively. Our aim is to find a coequalizer of φ_1 and φ_2, that is, an object Λ and a projection $\pi\colon \Delta \twoheadrightarrow \Lambda$ represented by

$$[[(b_1,U_{a_1}h_1,a_1),(b_2,U_{a_2}h_2,a_2),(b_3,U_{a_3}h_3,a_3),(b_4,U_{a_4}h_1,a_4)],[],[],[]]$$

such that $\pi \circ \varphi_1 = \pi \circ \varphi_2$. Visually speaking, the condition $\pi \circ \varphi_1 = \pi \circ \varphi_2$ means that in Figure 1.3 "taking the dotted arrows and then the solid ones" yields the same result as "taking the dashed arrows and then the solid ones". Of course, we could construct all sets and the maps explicitly, afterwards take the canonical coequalizer in **Sets** and finally define an action of G on it as in Proposition 1.1.14.

Algorithm 7: Equalizers in **Skeletal-G-Sets**

Input: morphisms $\varphi_1, \ldots, \varphi_s \colon \Omega \to \Delta$ in **Skeletal-G-Sets** and optionally a test object Λ' with a test morphism $\iota' \colon \Lambda' \to \Omega$

Output: an equalizer Λ of $\varphi_1, \ldots, \varphi_s$ with an embedding $i \colon \Lambda \to \Omega$; additionally the universal morphism $\psi \colon \Lambda' \to \Lambda$ if a test object is given

`EmbeddingOfEqualizer` and `Equalizer` (automatically derived by CAP):
let P be the empty set;
foreach *position p of* Ω **do**
    if *the components of the φ_t at position p are pairwise equal* **then**
        add p to P;
let ι be the embedding of the sub-G-set P of Ω;
let Λ be the source of ι;
`UniversalMorphismIntoEqualizer`:
if *a test object is given* **then**
    let ψ be the lift of ι' along the monomorphism ι;
return Λ and ι, and additionally ψ if a test object is given;

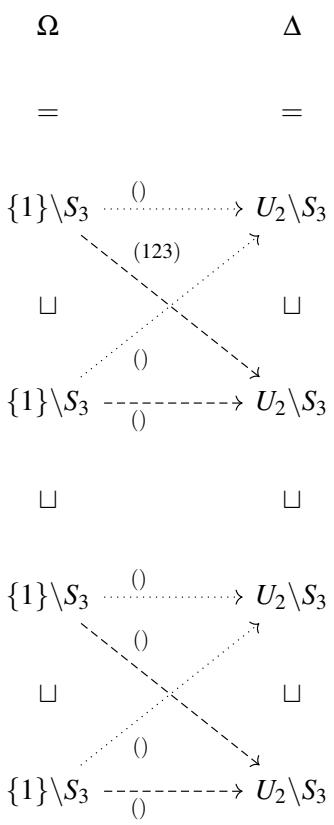


Figure 1.2: A visualization of the objects and maps in Example 1.1.38

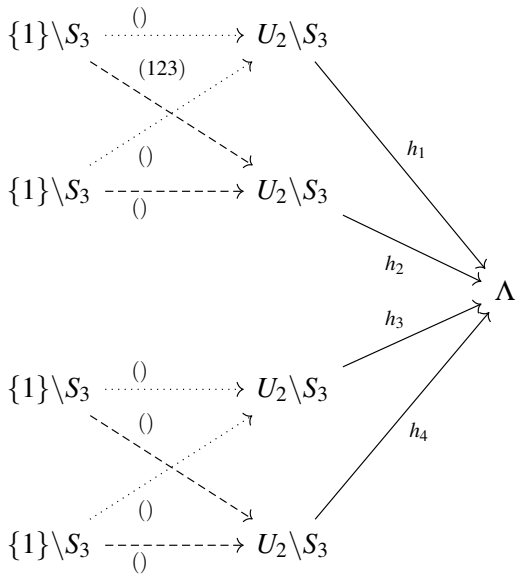


Figure 1.3: A visualization of the coequalizer in Example 1.1.38

However, we have implemented a more efficient way. The idea is to collect all necessary conditions which follow from the diagram in Figure 1.3 and then show that these conditions are already sufficient. The first observation is that the diagram in Figure 1.2 consists of two "connected components", the upper and the lower half. All components of π starting from a single connected component of the diagram (for example $(b_1, U_{a_1} h_1, a_1)$ and $(b_2, U_{a_2} h_2, a_2)$) have to have the same target position because otherwise "taking the dotted arrows" and "taking the dashed arrows" can never give the same result. For the converse, recall the definition of $\sim$ in Proposition 1.1.14: two elements of Δ are related by $\sim$ if and only if they are the images of a single $\omega \in \Omega$ under φ_1 and φ_2. But this implies that they lie in the same connected component of the diagram. Since taking the transitive hull of $\sim$ does not cross connected components, we do not want to relate elements of different connected components to each other. If two components of π have the same target position, this means that elements of the sources of the components are related. So components of π starting from different connected components of the diagram should not have the same target position. Thus, Λ must have at least two positions and the universal property implies that Λ is the "smallest" object with this property, so it should not have more. Therefore, we have $\Lambda = V_1\backslash S_3 \sqcup V_2\backslash S_3$ for suitable subgroups $V_1, V_2 \leq S_3$. This is visualized in Figure 1.4. Now, one can check that $V_1\backslash S_3$ and $V_2\backslash S_3$ are coequalizers of the upper connected component and the lower connected component respectively. Thus, we only have to compute coequalizers of individual connected components. We will make this precise in the following definition and lemma.

Definition 1.1.39 (Connected components of parallel maps)**.** Given two objects $\Omega = [m_1, \ldots, m_k]$ and $\Delta = [n_1, \ldots, n_k]$ in **Skeletal-G-Sets** and morphisms $\varphi_1, \ldots, \varphi_s \colon \Omega \to \Delta$ in **Skeletal-G-Sets** we define the set

$$R := \{(i, \ell_i, 1) \mid 1 \leq i \leq k \text{ and } 1 \leq \ell_i \leq m_i\} \cup \{(j, r_j, 2) \mid 1 \leq j \leq k \text{ and } 1 \leq r_j \leq n_j\}.$$

A triple $(i, \ell, 1) \in R$ corresponds to the position (i, ℓ) of Ω and a triple $(j, r, 2)$ corresponds to the position $(j, r) \in \Delta$. We define a relation on R: we set $(i, \ell, 1) \sim (j, r, 2)$ for $(i, \ell, 1), (j, r, 2) \in R$ if and

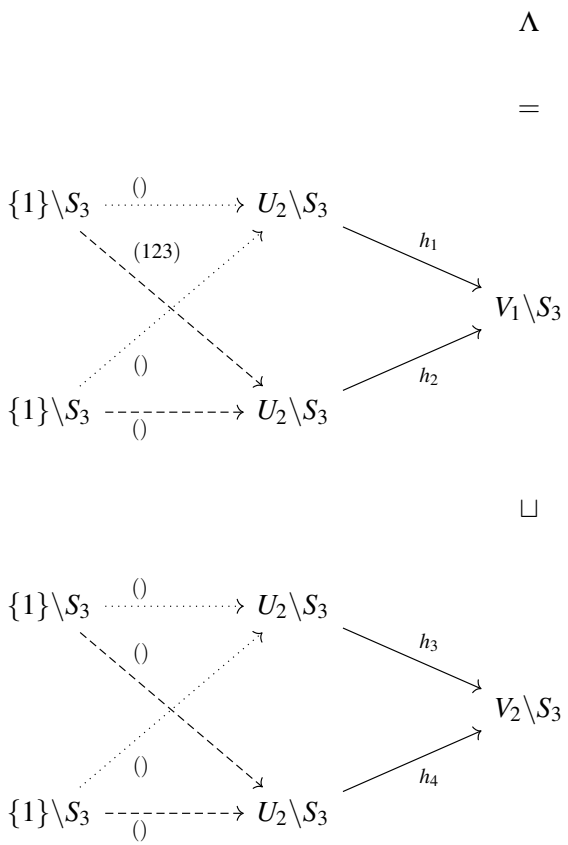


Figure 1.4: A refined visualization of the coequalizer in Example 1.1.38

only if there exists $t \in \{1,\dots,s\}$ such that the target position of the component of φ_t at position (i,ℓ) is (j,r). If this is the case, we say that $(i,\ell,1) \sim (j,r,2)$ is *induced* by φ_t. A *connected component* of $\varphi_1,\dots,\varphi_s$ is an equivalence class of the reflexive, symmetric and transitive hull of this relation.

Note that the components of morphisms correspond to edges in our diagrams and thus the connected components of $\varphi_1,\dots,\varphi_s$ are indeed the connected components of the corresponding diagram viewed as a graph.

Let C be a connected component of $\varphi_1,\dots,\varphi_s$. Then we define the *source* $S(C)$ of C as the sub-G-set of $R(\Omega)$ corresponding to the sub-G-set $\{(i,\ell) \mid (i,\ell,1) \in C\}$ of Ω, and the *target of a connected component* $T(C)$ of C as the sub-G-set of $R(\Delta)$ corresponding to the sub-G-set $\{(j,r) \mid (j,r,2) \in C\}$ of Δ. Let $\iota_S \colon S(C) \to R(\Omega)$ and $\iota_T \colon T(C) \to R(\Delta)$ be the corresponding embeddings. For $t \in \{1,\dots,s\}$ we define the *restriction* $\varphi_t|_C$ of φ_t to C as the lift of $\varphi_t \circ A(\iota_S)$ along the monomorphism $A(\iota_T)$. Note that this lift exists since by construction $R(\varphi_t) \circ \iota_S(S(C)) \subseteq \iota_T(T(C))$. A *coequalizer* of C is a coequalizer of the restrictions of $\varphi_1,\dots,\varphi_s$ to C.

Remark 1.1.40 (Functoriality of the coproduct in **Skeletal-G-Sets**)**.** For any $t \in \{1,\dots,s\}$ let $\varphi_t \colon \Omega_t \to \Delta_t$ be a morphism in **Skeletal-G-Sets**. Set $\Omega := \bigsqcup_t \Omega_t$ and $\Delta := \bigsqcup_t \Delta_t$. Then there exists a unique morphism $\varphi \colon \Omega \to \Delta$ such that for any $t \in \{1,\dots,s\}$ the following diagram commutes:

$$\begin{array}{ccc} \Omega_t & \xrightarrow{\varphi_t} & \Delta_t \\ {\scriptstyle \iota_{\Omega_t}}\downarrow & & \downarrow{\scriptstyle \iota_{\Delta_t}} \\ \Omega & \xrightarrow[\varphi]{} & \Delta \end{array}$$

To see this, note that for any $t \in \{1,\dots,s\}$ we have a morphism $\Omega_t \xrightarrow{\varphi_t} \Delta_t \xrightarrow{\iota_{\Delta_t}} \Delta$ and by the universal property of the coproduct, these morphisms induce a unique morphism $\Omega \to \Delta$. We write $\bigsqcup_t \varphi_t := \varphi$.

This is a special case of the functoriality of limits and colimits in arbitrary categories.

Lemma 1.1.41. *Let Ω and Δ be objects in* **Skeletal-G-Sets** *and let $\varphi_1,\dots,\varphi_s \colon \Omega \to \Delta$ be morphisms in* **Skeletal-G-Sets***. Then a coproduct of coequalizers of the connected components of $\varphi_1,\dots,\varphi_s$ is a coequalizer of $\varphi_1,\dots,\varphi_s$.*

Proof. Assume that we can decompose both $\Omega = \bigsqcup_{d=1}^{c} \Omega_d$ and $\Delta = \bigsqcup_{d=1}^{c} \Delta_d$ into sub-G-sets such that for any $t \in \{1,\dots,s\}$ and any $d \in \{1,\dots,c\}$ there exists a morphism $(\varphi_t)_d \colon \Omega_d \to \Delta_d$ such that the following diagram commutes:

$$\begin{array}{ccc} \Omega_d & \xrightarrow{(\varphi_t)_d} & \Delta_d \\ {\scriptstyle \iota_{\Omega_d}}\downarrow & & \downarrow{\scriptstyle \iota_{\Delta_d}} \\ \Omega & \xrightarrow{\varphi_t} & \Delta \end{array}$$

For any $d \in \{1,\dots,c\}$ let Λ_d be a coequalizer of $(\varphi_1)_d,\dots,(\varphi_s)_d$ and $\pi_d \colon \Delta_d \to \Lambda_d$ the corresponding projection. Set $\Lambda := \bigsqcup_d \Lambda_d$ and $\pi := \bigsqcup_d \pi_d$. Then we have the following commutative diagram:

$$\begin{array}{ccccc} \Omega_d & \xrightarrow{(\varphi_t)_d} & \Delta_d & \xrightarrow{\pi_d} & \Lambda_d \\ {\scriptstyle \iota_{\Omega_d}}\downarrow & & {\scriptstyle \iota_{\Delta_d}}\downarrow & & {\scriptstyle \iota_{\Lambda_d}}\downarrow \\ \Omega & \xrightarrow{\varphi_t} & \Delta & \xrightarrow{\pi} & \Lambda \end{array}$$

We want to show that Λ together with π is a coequalizer of $\varphi_1,\dots,\varphi_s$. First, we have to show that $\pi \circ \varphi_a = \pi \circ \varphi_b$ for all $a,b \in \{1,\dots,s\}$. But this can be checked on the sub-G-sets Ω_d of Ω and by

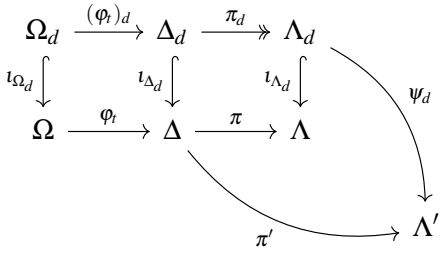
commutativity of the diagram we have

$$\pi \circ \varphi_a \circ \iota_{\Omega_d} = \iota_{\Lambda_d} \circ \pi_d \circ (\varphi_a)_d = \iota_{\Lambda_d} \circ \pi_d \circ (\varphi_b)_d = \pi \circ \varphi_b \circ \iota_{\Omega_d}.$$

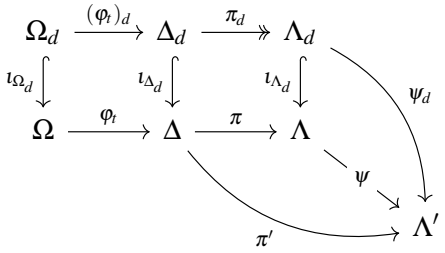
Now let $\pi' \colon \Delta \to \Lambda'$ be a morphism in **Skeletal-G-Sets** with $\pi' \circ \varphi_a = \pi' \circ \varphi_b$ for all $a, b \in \{1, \ldots, s\}$. Then for any $d \in \{1, \ldots, c\}$ and for all $a, b \in \{1, \ldots, s\}$ we have

$$\pi' \circ \iota_{\Delta_d} \circ (\varphi_a)_d = \pi' \circ \varphi_a \circ \iota_{\Omega_d} = \pi' \circ \varphi_b \circ \iota_{\Omega_d} = \pi' \circ \iota_{\Delta_d} \circ (\varphi_b)_d.$$

Thus, we get a unique morphism $\psi_d \colon \Lambda_d \to \Lambda'$ such that the following diagram commutes:

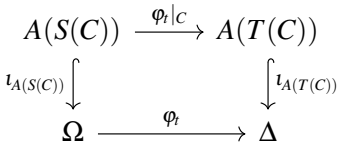


Finally, by the universal property of the coproduct we get a unique morphism $\psi \colon \Lambda \to \Lambda'$ such that the following diagram commutes:



This is a special case of the fact that taking colimits commutes with taking colimits.

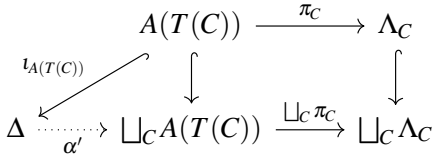
Now let $\mathscr{C}$ be the set of connected components of $\varphi_1, \ldots, \varphi_s$. Then one can easily check that $\Omega = \bigsqcup_{C \in \mathscr{C}} A(S(C))$ and $\Delta = \bigsqcup_{C \in \mathscr{C}} A(T(C))$, and that for any $C \in \mathscr{C}$ and any $t \in \{1, \ldots, s\}$ the following diagram commutes:



Thus, the decomposition into connected components has the properties which we have assumed at the beginning of the proof and the claim follows. □

Remark 1.1.42 (Implementation)**.** Let Ω and Δ be two objects in **Skeletal-G-Sets** and let $\varphi_1, \ldots, \varphi_s \colon \Omega \to \Delta$ be morphisms in **Skeletal-G-Sets**. The lemma above shows that if we can compute the projections π_C of coequalizers of the connected components of $\varphi_1, \ldots, \varphi_s$, we can afterwards take $\bigsqcup_C \pi_C$ to get the projection of a coequalizer of $\varphi_1, \ldots, \varphi_s$. Computing $\bigsqcup_C \pi_C$ can be done with the method `CoproductFunctorial` in CAP. However, one issue arises: the source of π is a coproduct of the $A(T(C))$ but in general this is not the same coproduct as Δ. Since we work in a skeletal setting, as

objects both agree, but in general the embeddings are different. Thus, the following diagram does *not* commute for $\alpha' = \mathrm{id}_\Delta$ because the triangle on the left does not commute:



However, we know that there exists an isomorphism $\alpha' \colon \Delta \to \bigsqcup_C A(T(C))$ which makes the diagram commute. If not concerned with the implementation, one would use the universal property of the coproduct Δ and choose α' to be the unique morphism $\Delta \to \bigsqcup_C A(T(C))$. However, in CAP we can only use the universal property of the coproduct with the embeddings given by `InjectionOfCofactorOfCoproduct`. But this is exactly what `CoproductFunctorial` uses and what we have denoted by $A(T(C)) \hookrightarrow \bigsqcup_C A(T(C))$ above. Thus, we can use the universal property of $\bigsqcup_C A(T(C))$ to get an isomorphism $\alpha \colon \bigsqcup_C A(T(C)) \to \Delta$ and choose $\alpha' \coloneqq \alpha^{-1}$. Thus, the projection $\pi \colon \Delta \to \bigsqcup_C \Lambda_C$ we are looking for is given by $\bigsqcup_C \pi_C \circ \alpha^{-1}$.

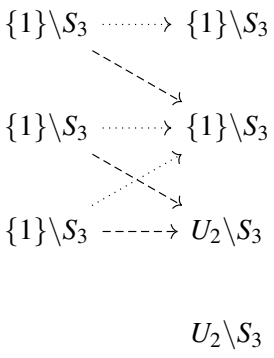
Example 1.1.43 (Finding connected components)**.** Choose $G = S_3$. Let $\Omega = [3,0,0,0]$ and $\Delta = [2,2,0,0]$ be objects in **Skeletal-G-Sets** and let $\varphi_1, \varphi_2 \colon \Omega \to \Delta$ be two morphisms in **Skeletal-G-Sets** represented by

$$[(1, U_1(), 1), (2, U_1(), 1), (2, U_1(), 1)], [], [], []$$

and

$$[(2, U_1(), 1), (1, U_2(), 2), (1, U_2(), 2)], [], [], []$$

respectively. We get the following diagram, where φ_1 corresponds to the dotted and φ_2 to the dashed arrows:



We want to find the connected components of φ_1 and φ_2. For this, we start with the position $(1,1)$ of Δ at, which corresponds to the element $(1,1,2)$ in R. Let $(i,\ell,1) \in R$ for R as in Definition 1.1.39. By the definition of $\sim$ in Definition 1.1.39, we have $(i,\ell,1) \sim (1,1,2)$ if and only if the target position of the component of φ_1 at position (i,ℓ) is $(1,1)$ or the target position of the component of φ_2 at position (i,ℓ) is $(1,1)$. Now, the positions (i,ℓ) with this property are exactly the preimage positions of the set $\{(1,1)\}$ under φ_1 or φ_2. There is only one such preimage position, namely the position $(1,1)$ of Ω, which corresponds to $(1,1,1) \in R$. Since we want to have equivalence classes of the transitive hull of $\sim$, we now have to continue by finding all triples $(j,r,2) \in R$ such that $(1,1,1) \sim (j,r,2)$. Again by definition of $\sim$, these correspond exactly to the image positions of the set $\{(1,1)\}$ under φ_1 or φ_2. These image positions are the positions $(1,1)$ and $(1,2)$ of Δ. Repeating this one more time, we get the preimage

positions $(1,1)$, $(1,2)$ and $(1,3)$ and the image positions $(1,1)$, $(1,2)$ and $(2,1)$. Note that repeating this again will not change the result anymore. Thus, we know that we have found all elements related to $(1,1,2)$ under the reflexive, symmetric and transitive hull of $\sim$, that is, a connected component C. The source $S(C)$ is the sub-G-set of $R(\Omega)$ corresponding to the sub-G-set $\{(1,1),(1,2),(1,3)\}$ of Ω and the target $T(C)$ is the sub-G-set of $R(\Delta)$ corresponding to the sub-G-set $\{(1,1),(1,2),(2,1)\}$ of Δ.

Now the only remaining position of Δ is the position $(2,2)$. There are no preimage positions of the set $\{(2,2)\}$ under φ_1 or φ_2. Thus, the equivalence class of $(2,2,2)$ only contains $(2,2,2)$ itself. Therefore, we have found another connected component C'. Its source $S(C')$ is the sub-G-set of $R(\Omega)$ corresponding to the sub-G-set $\varnothing$ of Ω and its target $T(C')$ is the sub-G-set of $R(\Delta)$ corresponding to the sub-G-set $\{(2,2)\}$ of Δ.

Every position of Ω is related to a position Δ, so it suffices to find the equivalence classes of the positions of Δ and hence we are done.

Example 1.1.44. We continue Example 1.1.38 and now only consider the upper connected component, that is, we restrict φ_1, φ_2 and π to $\Omega = [2,0,0,0]$ and $\Delta = [0,2,0,0]$. From the diagram we get the following conditions:

$$\begin{aligned} V_1 h_1() &= V_1 h_2(123), \\ V_1 h_1() &= V_1 h_2(). \end{aligned} \tag{1.1}$$

As always, V_1 is only determined up to conjugation. This is reflected in the fact that we can choose one h_t freely, since choosing it differently corresponds simply to taking a conjugate of V_1. Thus, we can fix $h_1 := ()$ and compute $h_2 = h_1()(123)^{-1} = (132)$ using the first equation considered as an equation in G, that is, not modulo V_1. Now we have found the group elements h_1 and h_2 in the definition of the projection π but we still have to determine V_1 such that π is well-defined and such that the second equation is fulfilled. The second equation gives us the condition $V_1() = V_1(132)$ which is equivalent to $(132) \in V_1$. The well-definedness of π gives us the following conditions:

$$\begin{aligned} \langle (23) \rangle = h_1 U_2 h_1^{-1} &\subseteq V_1, \\ \langle (12) \rangle = h_2 U_2 h_2^{-1} &\subseteq V_1. \end{aligned} \tag{1.2}$$

Thus, $\langle (23),(12),(132) \rangle \subseteq V_1$ which implies $V_1 = S_3$ and so the coequalizer $V_1 \backslash S_3$ must be the trivial G-set. To explain the general setting, let us assume that we do not know that the condition $\langle (23),(12),(132) \rangle \subseteq V_1$ already determines V_1. Then, we simply set $V_1 := \langle (23),(12),(132) \rangle$ and show that $V_1 \backslash S_3$ fulfills the properties of the coequalizer.

Note that we have constructed $V_1 \backslash S_3$ in **G-Sets**, not in **Skeletal-G-Sets**. To get an object in **Skeletal-G-Sets**, we have to find $g \in S_3$ such that $g V_1 g^{-1} = V \in \mathscr{U}$. Then $V_1 \backslash S_3 \cong V \backslash S_3$ and the tuple representing $V \backslash S_3$ can be read of immediately: if we write $V = U_i \in \mathscr{U}$, then the i-th entry of the tuple is 1 and all other entries are 0. An explicit isomorphism $V_1 \backslash S_3 \to V \backslash S_3$ is given by $V_1 h \mapsto V g h$. Thus, we only have to multiply the h_t with g to get the projection to $V \backslash S_3$ instead of $V_1 \backslash S_3$. Note that this corresponds to choosing $h_1 = g$ instead of $h_1 = ()$. In this example we could choose any $g \in S_3$ because we know that $V_1 = S_3$. Therefore, we stick to $g = ()$ and $V = V_1$.

To check that $V_1 \backslash S_3$ together with π is a coequalizer we could show that it is the canonical set-theoretic coequalizer. However, it is easier to check the properties of the coequalizer directly.

By (1.2), the projection π is well-defined and by (1.1) we have $\pi \circ f_1 = \pi \circ f_2$.

Now, let Λ' be an object in **Skeletal-G-Sets** and $\pi' \colon \Delta \to \Lambda'$ a morphism given by

$$[[(b'_1, U_{a'_1} h'_1, a'_1), (b'_2, U_{a'_2} h'_2, a'_2)], [], [], []]$$

such that $\pi' \circ f_1 = \pi' \circ f_2$. We have to show that there exists a unique morphism $\psi\colon \Lambda \to \Lambda'$ with the single component $(b, U_a g_\psi, a)$ such that $\psi \circ \pi = \pi'$. By the same arguments as in Example 1.1.38, both components of π' and the unique component of ψ must have the same target position. So let $V' := U_{a'_1} = U_{a'_2} = U_a$.

The equality $\psi \circ \pi = \pi'$ is equivalent to the conditions $V' g_\psi h_1 = V' h'_1$ and $V' g_\psi h_2 = V' h'_2$. Since we have fixed $h_1 = ()$, the first condition implies $V' g_\psi = V' h'_1$. Thus, $(b, U_a g_\psi, a)$ is uniquely defined. Setting $U_a g_\psi := V' h'_1$ also fulfills the second condition because we have

$$\begin{aligned} V' g_\psi h_2 &= V' h'_1 h_2 && (1.3)\\ &= V' h'_1 h_1 () (123)^{-1} && (1.4)\\ &= V' h'_1 () (123)^{-1} && (1.5)\\ &= V' h'_2. && (1.6) \end{aligned}$$

For the equality in (1.4) we use that we have computed $h_2 = h_1()(123)^{-1}$ above. For the equality in (1.6) we use the equation $V' h'_1() = V' h'_2 (123)$ following from the assumption $h' \circ f_1 = h' \circ f_2$.

Finally, we have to show that ψ is in fact well-defined. This comes down to showing three inclusions:

$$\begin{aligned} g_\psi \langle (23) \rangle g_\psi^{-1} &\subseteq V', && (1.7)\\ g_\psi \langle (12) \rangle g_\psi^{-1} &\subseteq V', && (1.8)\\ g_\psi \langle (132) \rangle g_\psi^{-1} &\subseteq V'. && (1.9) \end{aligned}$$

The inclusion in (1.7) is just $h'_1 \langle (23) \rangle {h'_1}^{-1} \subseteq V'$, which holds true because h' is well-defined. For the inclusion in (1.8), recall that the subgroup $\langle (12) \rangle$ was originally given as $h_2 \langle (23) \rangle h_2^{-1}$ in (1.2), so we can rewrite the inclusion as $h'_1 h_2 \langle (23) \rangle h_2^{-1} {h'_1}^{-1} \subseteq V'$. Using the equality $V' h'_1 h_2 = V' h'_2$ (see (1.3) and (1.6)), this in turn is equivalent to $h'_2 \langle (23) \rangle {h'_2}^{-1} \subseteq V'$, which again holds true because h' is well-defined. Finally, the inclusion in (1.9) holds true because the assumption $h' \circ f_1 = h' \circ f_2$ implies $V'() = V'(132)$.

We make the algorithm precise in Algorithm 8 and Algorithm 9.

Proposition 1.1.45. *Algorithm 8 terminates and yields the correct result.*

Proof. For any position p of Ω and $t \in \{1, \dots, s\}$ let $((r_t)_p, U_{(j_t)_p}(g_t)_p, (j_t)_p)$ be the component of φ_t at position p. Whenever we implicitly choose a representative of some $U_{(j_t)_p}(g_t)_p$ during the proof, let this representative be $(g_t)_p$. See Remark 1.1.46 for more details on this.

We first show that the algorithm terminates, that is, that solutions for all $h_{(j,r)}$ are found in the loop in line 11 after finitely many iterations. For this, consider one variable $h_{(j,r)}$. Since $\varphi_1, \dots, \varphi_s$ have only a single connected component, $(j', r', 2)$ and $(j, r, 2)$ lie in the same connected component, where (j', r') is the position of Δ fixed in the algorithm. By definition, this means that there exists a chain

$$(j', r', 2) =: (j_1, r_1, 2) \sim (i_1, \ell_1, 1) \sim (j_2, r_2, 2) \sim \cdots \sim (i_c, \ell_c, 1) \sim (j_{c+1}, r_{c+1}, 2) := (j, r) \qquad (1.10)$$

where we have replaced $\sim$ with its own symmetric hull for simpler notation. Now, we consider a triple in this chain, that is, we fix $d \in \{1, \dots, c\}$ and look at

$$(j_d, r_d, 2) \sim (i_d, \ell_d, 1) \sim (j_{d+1}, r_{d+1}, 2).$$

Choose $a \in \{1, \dots, s\}$ such that $(i_d, \ell_d, 1) \sim (j_d, r_d, 2)$ is induced by φ_a and choose $b \in \{1, \dots, s\}$ such that $(i_d, \ell_d, 1) \sim (j_{d+1}, r_{d+1}, 2)$ is induced by φ_b. When building the system of equations we add the

Algorithm 8: `ProjectionOntoCoequalizerOfAConnectedComponent`

Input: morphisms $\varphi_1, \ldots, \varphi_s \colon \Omega \to \Delta$ in **Skeletal-G-Sets** which have only a single connected component

Output: a coequalizer Λ of $\varphi_1, \ldots, \varphi_s$ and its projection $\pi \colon \Delta \to \Lambda$

Building the system of equations:
for $a, b \in \{1, \ldots, s\}$ **do**
    Note: Since for $a = b$ the equations are trivially fulfilled and since the order of a and b is irrelevant, we can actually restrict to the cases where $a < b$.
    foreach *position* (i, ℓ) *of* Ω **do**
        let $(r_a, U_a g_a, j_a)$ be the the component of φ_a at position (i, ℓ);
        let $(r_b, U_b g_b, j_b)$ be the the component of φ_b at position (i, ℓ);
        add the equation $h_{(j_a, r_a)} \cdot g_a = h_{(j_b, r_b)} \cdot g_b$ with variables $h_{(j_a, r_a)}$ and $h_{(j_b, r_b)}$ to the system of equations;
Solving the system of equations:
let (j', r') be a fixed position of Δ;
set $h_{(j', r')} := 1 \in G$;
repeat
    foreach *equation* $h_{(j_a, r_a)} \cdot g_a = h_{(j_b, r_b)} \cdot g_b$ **do**
        if *the value of* $h_{(j_a, r_a)}$ *is unknown and the value of* $h_{(j_b, r_b)}$ *is known* **then**
            solve the equation for $h_{(j_a, r_a)}$, that is, set $h_{(j_a, r_a)} := h_{(j_b, r_b)} \cdot g_b \cdot g_a^{-1}$;
        if *the value of* $h_{(j_a, r_a)}$ *is known and the value of* $h_{(j_b, r_b)}$ *is unknown* **then**
            solve the equation for $h_{(j_b, r_b)}$, that is, set $h_{(j_b, r_b)} := h_{(j_a, r_a)} \cdot g_a \cdot g_b^{-1}$;
until *all variables* $h_{(j,r)}$ *have known values*;
Construct Λ and π:
let X be the empty set;
foreach *position* (j, r) *of* Δ **do**
    add the elements of $h_{(j,r)} U_j h_{(j,r)}^{-1}$ to X;
foreach *equation* $h_{(j_a, r_a)} \cdot g_a = h_{(j_b, r_b)} \cdot g_b$ **do**
    add $h_{(j_b, r_b)} \cdot g_b \cdot g_a^{-1} \cdot h_{(j_a, r_a)}^{-1}$ to X;
let $V := \langle X \rangle \leq G$;
find $g \in G$ such that $gVg^{-1} = U_{i'} \in \mathscr{U}$;
set $\Lambda := A(U_{i'} \backslash G)$;
let π be the morphism $\Delta \to \Lambda$ which has component $(1, U_{i'} g h_{(j,r)}, i')$ at position (j, r);
return Λ and π;

equation $h_{(j_d,r_d)} \cdot (g_a)_{(i_d,\ell_d)} = h_{(j_{d+1},r_{d+1})} \cdot (g_b)_{(i_d,\ell_d)}$. In particular, if the value of $h_{(j_d,r_d)}$ is known we can solve the equation for $h_{(j_{d+1},r_{d+1})}$, that is, we set

$$h_{(j_{d+1},r_{d+1})} := h_{(j_d,r_d)} \cdot (g_a)_{(i_d,\ell_d)} \cdot (g_b)^{-1}_{(i_d,\ell_d)}. \tag{1.11}$$

Since the chain is finite and since we have fixed a value for $h_{(j',r')}$, after finitely many steps we will find a solution for $h_{(j,r)}$.

Next, we show that the projection π is well-defined. Choose g and i' as in the algorithm. Then the component of π at a position (j,r) of Δ is $(1, U_{i'} g h_{(j,r)}, i')$. We have

$$g h_{(j,r)} U_j (g h_{(j,r)})^{-1} = g h_{(j,r)} U_j h^{-1}_{(j,r)} g^{-1} \subseteq g V g^{-1} = U_{i'}$$

by line 21 and thus, π is well-defined.

Next, we show that $\pi \circ \varphi_a = \pi \circ \varphi_b$ for all $a,b \in \{1,\dots,s\}$. This is equivalent to all components being equal. Let p be a position of Ω, let $(r_a, U_{j_a} g_a, j_a)$ be the component of φ_a at position p and let $(r_b, U_{j_b} g_b, j_b)$ be the component of φ_b at position p. We have to show that $U_{i'} g h_{(j_a,r_a)} g_a = U_{i'} g h_{(j_b,r_b)} g_b$, which is equivalent to $g h_{(j_b,r_b)} g_b g_a^{-1} h^{-1}_{(j_a,r_a)} g^{-1} \in U_{i'}$. By line 23 we have $h_{(j_b,r_b)} g_b g_a^{-1} h^{-1}_{(j_a,r_a)} \in V$ and the claim follows by using $g V g^{-1} = U_{i'}$.

Now, let Λ' be a test object with $\pi' \colon \Delta \to \Lambda'$ such that $\pi' \circ \varphi_a = \pi' \circ \varphi_b$ for all $a,b \in \{1,\dots,s\}$. For any position p of Δ let $(y_p, U_{x_p} h'_p, x_p)$ be the component of π' at position p. Let $\psi \colon \Lambda \to \Lambda'$ be the morphism in **Skeletal-G-Sets** with the single component $(y_{(j',r')}, U_{x_{(j',r')}} h'_{(j',r')} g^{-1}, x_{(j',r')})$. Before we prove that ψ is well-defined, we first prove the following claim: we have

$$V' h'_{(j',r')} h_{(j,r)} = V' h'_{(j,r)} \tag{1.12}$$

for all positions (j,r) of Δ. This follows if we consider the same chain as in (1.10) and, again, a triple

$$(j_d, r_d, 2) \sim (i_d, \ell_d, 1) \sim (j_{d+1}, r_{d+1}, 2)$$

in this chain. Let $a,b \in \{1,\dots,s\}$ such that φ_a induces $(i_d,\ell_d,1) \sim (j_d,r_d,2)$ and φ_b induces $(i_d,\ell_d,1) \sim (j_{d+1},r_{d+1},2)$. Then by (1.11) we have

$$h_{(j_d,r_d)} \cdot (g_a)_{(i_d,\ell_d)} = h_{(j_{d+1},r_{d+1})} \cdot (g_b)_{(i_d,\ell_d)}. \tag{1.13}$$

Additionally, $\pi' \circ \varphi_a = \pi' \circ \varphi_b$ implies $x_{(j_d,r_d)} = x_{(j_{d+1},r_{d+1})}$, $y_{(j_d,r_d)} = y_{(j_{d+1},r_{d+1})}$, and

$$U_{x_{(j_d,r_d)}} h'_{(j_d,r_d)} (g_a)_{(i_d,\ell_d)} = U_{x_{(j_{d+1},r_{d+1})}} h'_{(j_{d+1},r_{d+1})} (g_b)_{(i_d,\ell_d)}. \tag{1.14}$$

For $V' := U_{x_{(j_d,r_d)}} = U_{x_{(j_{d+1},r_{d+1})}}$, combining (1.13) and (1.14) gives

$$V' h'_{(j_d,r_d)} h^{-1}_{(j_d,r_d)} h_{(j_{d+1},r_{d+1})} = V' h'_{(j_{d+1},r_{d+1})}.$$

Iterating this, we get $U_{x_{(j_d,r_d)}} = V'$ for all $d \in \{1,\dots,(c+1)\}$ and

$$\begin{aligned}
V' h'_{(j,r)} = V' h'_{(j_{c+1},r_{c+1})} &= V' h'_{(j_c,r_c)} h^{-1}_{(j_c,r_c)} h_{(j_{c+1},r_{c+1})} \\
&= V' h'_{(j_{c-1},r_{c-1})} h^{-1}_{(j_{c-1},r_{c-1})} h_{(j_c,r_c)} h^{-1}_{(j_c,r_c)} h_{(j_{c+1},r_{c+1})} \\
&= V' h'_{(j_{c-1},r_{c-1})} h^{-1}_{(j_{c-1},r_{c-1})} h_{(j_{c+1},r_{c+1})} \\
&\vdots \\
&= V' h'_{(j_1,r_1)} h^{-1}_{(j_1,r_1)} h_{(j_{c+1},r_{c+1})} \\
&= V' h'_{(j_1,r_1)} h_{(j_{c+1},r_{c+1})} = V' h'_{(j',r')} h_{(j,r)}.
\end{aligned}$$

This is what we have claimed. In particular, there exists $v'_{(j,r)} \in V'$ such that

$$v'_{(j,r)} h'_{(j,r)} = h'_{(j',r')} h_{(j,r)}. \tag{1.15}$$

Now, we show that ψ is well-defined, that is, that we have $h'_{(j',r')} g^{-1} U_{i'} (h'_{(j',r')} g^{-1})^{-1} \subseteq V'$. This is equivalent to

$$h'_{(j',r')} V {h'_{(j',r')}}^{-1} \subseteq V'.$$

Since V was defined as a generated subgroup, it suffices to show this inclusion for the generating sets instead of V. That is, we have to show

$$h'_{(j',r')} h_{(j,r)} U_j h^{-1}_{(j,r)} {h'_{(j',r')}}^{-1} \subseteq V' \tag{1.16}$$

for each position (j,r) of Δ and

$$h'_{(j',r')} h_{(j_b,r_b)} \cdot g_b \cdot g_a^{-1} \cdot h^{-1}_{(j_a,r_a)} {h'_{(j',r')}}^{-1} \in V' \tag{1.17}$$

for each equation $h_{(j_a,r_a)} \cdot g_a = h_{(j_b,r_b)} \cdot g_b$. Using (1.15), we can write (1.16) as

$$v'_{(j,r)} h'_{(j,r)} U_j {h'_{(j,r)}}^{-1} {v'_{(j,r)}}^{-1} \subseteq V',$$

which holds true since π' is well-defined. Analogously, we can write (1.17) as

$$v'_{(j_b,r_b)} h'_{(j_b,r_b)} \cdot g_b \cdot g_a^{-1} \cdot {h'_{(j_a,r_a)}}^{-1} {v'_{(j_a,r_a)}}^{-1} \in V'. \tag{1.18}$$

By construction of the system of equations, there exists a position (i,ℓ) of Ω such that the component of φ_a at position (i,ℓ) is $(r_a, U_a g_a, j_a)$ and the component of φ_b at position (i,ℓ) is $(r_b, U_b g_b, j_b)$. Thus, $\pi' \circ \varphi_a = \pi' \circ \varphi_b$ implies that $V' h'_{(j_a,r_a)} \cdot g_a = V' h'_{(j_b,r_b)} \cdot g_b$ and therefore (1.18) holds true.

Next, we show that $\psi \circ \pi = \pi'$. This is equivalent to all components being equal, that is, $V' h'_{(j',r')} g^{-1} g h_{(j,r)} = V' h'_{(j,r)}$ for all positions (j,r) of Δ, and this is just (1.12).

In particular, ψ is the colift of π' along the epimorphism π. Such a colift is unique if it exists. Thus, ψ is uniquely defined and this finishes the proof. □

Remark 1.1.46. The intermediate results of the proof are certainly neither independent of the choices of the representatives at the beginning of the proof, nor are they independent of the choice of the chain (1.10). Concretely, the left hand side of (1.11) depends both on the choice of the g_a and g_b and on the choice of the chain. However, this is no surprise because the coequalizer together with the projection is only determined up to isomorphism. Mathematically, this is not a problem because in the proof we fix representatives and a chain and can thus use (1.13) as a consequence of (1.11). In the implementation, we fix representatives when building the system of equations and implicitly build the chain when solving the system of equations. If we would do this both for computing a coequalizer and for computing the universal morphism, we would have to ensure that both implementations are consistent. However, we will compute the universal morphism as a colift along an epimorphism. Thus, at no point inconsistencies can arise.

Proposition 1.1.47. *Algorithm 9 terminates and yields the correct result.*

Algorithm 9: Coequalizers in **Skeletal-G-Sets**

Input: two objects Ω and Δ in **Skeletal-G-Sets**, morphisms $\varphi_1, \ldots, \varphi_s \colon \Omega \to \Delta$ in **Skeletal-G-Sets**, optionally a test object Λ' with a test morphism $\pi' \colon \Lambda' \to \Omega$

Output: a coequalizer Λ of $\varphi_1, \ldots, \varphi_s$ with a projection $\pi \colon \Delta \to \Lambda$; if a test object is given additionally the universal morphism $\psi \colon \Lambda \to \Lambda'$

`ProjectionOntoCoequalizer` and `Coequalizer` (automatically derived by CAP):
let P the set of positions of Δ;
while *P is not the empty set* **do**
    let p be an element of P;
    Find the connected component containing p:
    let $I := \{p\}$;
    repeat
        let L be the union of the preimage positions of I under the φ_t;
        if *L is non-empty* **then**
            let I be the union of the image positions of the φ_t restricted to the sub-G-set L of Ω;
    until *I has stabilized, that is, it has not changed during the last iteration*;
    Compute a coequalizer of the connected component:
    let C be the connected component of $\varphi_1, \ldots, \varphi_s$ with $S(C)$ corresponding to the sub-G-set L of Ω and $T(C)$ corresponding to the sub-G-set I of Δ;
    compute a coequalizer Λ_C of $\varphi_1|_C, \ldots, \varphi_s|_C$ and its projection π_C;
    let ι_C be the embedding of the sub-G-set I of Δ;
    set $P := P \setminus I$;
Take the coproduct:
let $\tau := \bigsqcup_C \pi_C \colon \bigsqcup_C A(T(C)) \to \bigsqcup_C \Lambda_C =: \Lambda$;
let α be the unique morphism from $\bigsqcup_C A(T(C))$ to Δ induced by the embeddings $\iota_C \colon A(T(C)) \hookrightarrow \Delta$;
set $\pi := \tau \circ \alpha^{-1}$;
`UniversalMorphismFromCoequalizer`:
if *a test object is given* **then**
    let ψ be the colift of π' along the epimorphism π;
return Λ and π, and additionally ψ if a test object is given;

Proof. We first check that the algorithm terminates, that is, that the loops in lines 3 and 7 terminate after finitely many steps. We start with the loop in line 7. If L is empty, we do not change I and thus exit the loop. If L is non-empty, the size of I cannot get smaller because taking preimage positions of a set and afterwards the image positions of these is monotonically increasing. Since the set of positions of Δ is finite, I cannot grow infinitely and thus gets stationary. Then, we exit the loop. Additionally, this shows that after the loop, I cannot be empty. Thus, at the end of the loop in line 3, we remove at least one element from the finite set P. Therefore, after finitely many iterations, P is empty and we exit the loop. For the explanation why the loop in line 3 indeed loops over all connected components, see Example 1.1.43. For the correctness of the rest of the algorithm see Remark 1.1.42 and Proposition 1.1.14. □

1.2 The Tannaka Reconstruction Theorem

In this section, we will state a general version of the Tannaka Reconstruction Theorem and afterwards give a proof in our setting. We will not define all the notions in the general version. For details see [Tan]. Afterwards, we show that the results carry over if we replace **G-Sets** by a category $\mathscr{C}$ which is equivalent to **G-Sets** in a suitable way. We start by defining some notions and proving some results which we will need later.

Remark 1.2.1 (Opposite monoids and anti-isomorphisms)**.** To any monoid $M = (M, \cdot)$ we can associate the *opposite monoid* $M^{\mathrm{op}} := (M, *)$ with opposite multiplication $m * n := n \cdot m$ for all $m, n \in M$. In this setting it makes sense to define *anti-isomorphisms*, that is, bijective maps $\varphi\colon M \to N$ between monoids such that $\varphi(m \cdot n) = \varphi(n) \cdot \varphi(m)$ for all $m, n \in M$. Then the identity on sets $\mathrm{id}\colon M \to M^{\mathrm{op}}$ is an anti-isomorphism. Note that if we can invert elements, that is, if $M = G$ is a group, then inverting elements is an anti-isomorphism $G \to G$ respectively an isomorphism $G \to G^{\mathrm{op}}$. Since the composition of two anti-isomorphisms is an isomorphism, by composing with the inversion we can make any anti-isomorphism $M \to N$ of monoids into an isomorphism if M or N is a group. In particular, if we want to show that a group and a monoid are isomorphic as monoids, it suffices to find an anti-isomorphism between them. Additionally, if a group G and a monoid M are isomorphic as monoids, then M is already a group and G and M are isomorphic as groups. To check this, let $\varphi\colon G \to M$ be an isomorphism of monoids. For $m \in M$ define $m' := \varphi((\varphi^{-1}(m))^{-1})$. Then we have:

$$\begin{aligned} m'm &= \varphi((\varphi^{-1}(m))^{-1})m \\ &= \varphi((\varphi^{-1}(m))^{-1})\varphi(\varphi^{-1}(m)) \\ &= \varphi((\varphi^{-1}(m))^{-1}\varphi^{-1}(m)) \\ &= \varphi(1) \\ &= 1 \end{aligned}$$

and analogously $mm' = 1$. Thus, m' is an inverse of m, M is a group, and G and M are isomorphic as groups via φ.

Remark 1.2.2 (Natural transformations as elements of a product)**.** Let $\mathscr{C}$ and $\mathscr{D}$ be categories and let $F, F'\colon \mathscr{C} \to \mathscr{D}$ be functors. A natural transformation $\alpha\colon F \to F'$ is given by components α_C for all objects C in $\mathscr{C}$ where $\alpha_C\colon F(C) \to F'(C)$ is a morphism in $\mathscr{D}$. Thus, we can think of α as the tuple $(\alpha_C)_{C \in \mathrm{Obj}(\mathscr{C})} \in \prod_{C \in \mathrm{Obj}(\mathscr{C})} \mathrm{Hom}_{\mathscr{D}}(F(C), F'(C))$. Conversely, let $(\alpha_C)_{C \in \mathrm{Obj}(\mathscr{C})} \in \prod_{C \in \mathrm{Obj}(\mathscr{C})} \mathrm{Hom}_{\mathscr{D}}(F(C), F'(C))$. Then we can check if for all $C_1, C_2 \in \mathrm{Obj}(\mathscr{C})$ and $\varphi \in \mathrm{Hom}_{\mathscr{C}}(C_1, C_2)$ the following diagram commutes:

$$\begin{array}{ccc} F(C_1) & \xrightarrow{F(\varphi)} & F(C_2) \\ \downarrow{\scriptstyle \alpha_{C_1}} & & \downarrow{\scriptstyle \alpha_{C_2}} \\ F'(C_1) & \xrightarrow{F'(\varphi)} & F'(C_2) \end{array}$$

If this is the case, then $(\alpha_C)_{C \in \mathrm{Obj}(\mathscr{C})}$ defines a natural transformation $\alpha\colon F \to F'$ with components α_C. In this sense, the class of natural transformations $F \to F'$ is a subclass of $\prod_{C \in \mathrm{Obj}(\mathscr{C})} \mathrm{Hom}_{\mathscr{D}}(F(C), F'(C))$.

Lemma 1.2.3. *Let $\mathscr{C}$ be a category and $A \cong B$ isomorphic objects in $\mathscr{C}$. Then $\mathrm{Hom}_{\mathscr{C}}(A, A)$ and $\mathrm{Hom}_{\mathscr{C}}(B, B)$ are isomorphic as monoids with regard to the composition of morphisms.*

Proof. Let $\varphi\colon A \to B$ be an isomorphism. This isomorphism induces the map

$$f\colon \operatorname{Hom}_{\mathscr{C}}(A,A) \to \operatorname{Hom}_{\mathscr{C}}(B,B)$$
$$\tau \mapsto \varphi \circ \tau \circ \varphi^{-1}$$

which makes the following diagram commute:

$$\begin{array}{ccc} A & \xrightarrow{\tau} & A \\ {\scriptstyle\varphi}\downarrow & & \downarrow{\scriptstyle\varphi} \\ B & \xrightarrow[f(\tau)]{} & B \end{array}$$

Then for all $\tau_1, \tau_2 \in \operatorname{Hom}_{\mathscr{C}}(A,A)$ we have

$$f(\tau_1 \circ \tau_2) = \varphi \circ \tau_1 \circ \tau_2 \circ \varphi^{-1} = \varphi \circ \tau_1 \circ \varphi^{-1} \circ \varphi \circ \tau_2 \circ \varphi^{-1} = f(\tau_1) \circ f(\tau_2),$$

that is, f is an isomorphism of monoids. □

For the statement of the theorem we need the notion of an *end* of a functor. An end can be defined in many equivalent ways. We will give the definition as an equalizer in enriched category theory, which fits best into our setting.

Definition 1.2.4 (End [end])**.** Let V be a symmetric monoidal category, $\mathscr{C}$ a V-enriched category and $F'\colon \mathscr{C}^{\mathrm{op}} \times \mathscr{C} \to V$ a V-enriched functor. The *end* of F' is the equalizer $\operatorname{End}(F')$ of the morphisms ρ and λ defined as follows:

$$\rho, \lambda : \prod_{C \in \operatorname{Obj}(\mathscr{C})} F'(C,C) \longrightarrow \prod_{C_1, C_2 \in \operatorname{Obj}(\mathscr{C})} [\operatorname{Hom}_{\mathscr{C}}(C_1,C_2), F'(C_1,C_2)]$$

where $[-,-]$ is the internal hom in V and the components of ρ and λ are given by:

$$\rho_{C_1,C_2} : \prod_{C \in \operatorname{Obj}(\mathscr{C})} F'(C,C) \xrightarrow{\pi_{C_1}} F'(C_1,C_1) \longrightarrow [\operatorname{Hom}_{\mathscr{C}}(C_1,C_2), F'(C_1,C_2)]$$

where the second arrow is the adjunct of $F'(C_1,-)\colon \operatorname{Hom}_{\mathscr{C}}(C_1,C_2) \to [F'(C_1,C_1), F'(C_1,C_2)]$ and

$$\lambda_{C_1,C_2} : \prod_{C \in \operatorname{Obj}(\mathscr{C})} F'(C,C) \xrightarrow{\pi_{C_2}} F'(C_2,C_2) \longrightarrow [\operatorname{Hom}_{\mathscr{C}}(C_1,C_2), F'(C_1,C_2)]$$

where the second arrow is the adjunct of $F'(-,C_2)\colon \operatorname{Hom}_{\mathscr{C}}(C_1,C_2) \to [F'(C_2,C_2), F'(C_1,C_2)]$. For a functor $F\colon \mathscr{C} \to V$ we define the *end* $\operatorname{End}(F) := \operatorname{End}(\operatorname{Hom}_V(F(-),F(-)))$.

Theorem 1.2.5 (Tannaka Reconstruction Theorem [Tan])**.** *Let V be a locally small closed symmetric monoidal category with all limits and A a monoid in V. Let* **A-Mod** *be the V-enriched category of all A-modules in V and let $F\colon$* **A-Mod** *$\to V$ be the forgetful fiber functor. Then $A \cong \operatorname{End}(F)$, that is, we can reconstruct A from the category* **A-Mod** *and the forgetful fiber functor F.*

In our setting we choose $V =$ **Sets** and $A = G$ to be a finite group. The category **A-Mod** is just **G-Sets** and $F = U\colon$ **G-Sets** $\to$ **Sets** is the forgetful functor. We first show that in this setting $\operatorname{End}(U)$ coincides with the class of all natural transformations $U \to U$ and afterwards phrase and prove the theorem in our setting.

Proposition 1.2.6. *Let $\mathscr{C}$ be a category and $F\colon \mathscr{C} \to \mathbf{Sets}$ a functor. For $C \in \mathrm{Obj}(\mathscr{C})$ set*

$$S_C := \mathrm{Hom}_{\mathbf{Sets}}(F(C), F(C))$$

and for $C_1, C_2 \in \mathrm{Obj}(\mathscr{C})$ set

$$T_{C_1,C_2} := \mathrm{Hom}_{\mathbf{Sets}}(\mathrm{Hom}_{\mathscr{C}}(C_1,C_2), \mathrm{Hom}_{\mathbf{Sets}}(F(C_1),F(C_2))).$$

We set

$$S := \prod_{C \in \mathrm{Obj}(\mathscr{C})} S_C$$

and

$$T := \prod_{C_1,C_2 \in \mathrm{Obj}(\mathscr{C})} T_{C_1,C_2}.$$

Let $\rho\colon S \to T$ be the unique morphism in **Sets** *with components given by*

$$\rho_{C_1,C_2}\colon S \xrightarrow{\pi_{C_1}} S_{C_1} \to T_{C_1,C_2}$$

where the second arrow maps $f \in \mathrm{Hom}_{\mathbf{Sets}}(F(C_1),F(C_1))$ to the map

$$\begin{aligned} \mathrm{Hom}_{\mathscr{C}}(C_1,C_2) &\to \mathrm{Hom}_{\mathbf{Sets}}(F(C_1),F(C_2))) \\ \varphi &\mapsto F(\varphi) \circ f \end{aligned}$$

Analogously, let $\lambda : S \to T$ be the unique morphism in **Sets** *with components given by*

$$\lambda_{C_1,C_2}\colon S \xrightarrow{\pi_{C_2}} S_{C_2} \to T_{C_1,C_2}$$

where the second arrow maps $f \in \mathrm{Hom}_{\mathbf{Sets}}(F(C_2),F(C_2))$ to the map

$$\begin{aligned} \mathrm{Hom}_{\mathscr{C}}(C_1,C_2) &\to \mathrm{Hom}_{\mathbf{Sets}}(F(C_1),F(C_2))) \\ \varphi &\mapsto f \circ F(\varphi) \end{aligned}$$

Then the equalizer E of ρ and λ is the class of all natural transformations $F \to F$, that is, $E = \mathrm{Hom}(F,F)$.

Proof. Let $\alpha = (\alpha_C)_{C \in \mathrm{Obj}(\mathscr{C})} \in S$. Then α is an element of E if and only if $\rho(\alpha) = \lambda(\alpha)$. By the universal property of the product this is equivalent to $\rho_{C_1,C_2}(\alpha) = \lambda_{C_1,C_2}(\alpha)$ for all $C_1, C_2 \in \mathrm{Obj}(\mathscr{C})$. Since $\rho_{C_1,C_2}(\alpha)$ and $\lambda_{C_1,C_2}(\alpha)$ are elements of T_{C_1,C_2}, that is, maps themselves, they are equal if and only if $\rho_{C_1,C_2}(\alpha)(\varphi) = \lambda_{C_1,C_2}(\alpha)(\varphi)$ for all $\varphi \in \mathrm{Hom}_{\mathscr{C}}(C_1,C_2)$. By the definition of ρ_{C_1,C_2} and λ_{C_1,C_2}, this is equivalent to $F(\varphi) \circ \alpha_{C_1} = \alpha_{C_2} \circ F(\varphi)$ for all $\varphi \in \mathrm{Hom}_{\mathscr{C}}(C_1,C_2)$.

Summing up, we get that $\rho(\alpha) = \lambda(\alpha)$ if and only if $F(\varphi) \circ \alpha_{C_1} = \alpha_{C_2} \circ F(\varphi)$ for all $C_1, C_2 \in \mathrm{Obj}(\mathscr{C})$ and $\varphi \in \mathrm{Hom}_{\mathscr{C}}(C_1,C_2)$. This is exactly the definition of a natural transformation $F \to F$. □

Remark 1.2.7 (Redefinition of the end)**.** We have not shown that the assumptions in Proposition 1.2.6 match the definition of the end applied to the case $V = \mathbf{Sets}$. Thus, for completeness we redefine the end in the case $V = \mathbf{Sets}$: Let $\mathscr{C}$ be a category and $F\colon \mathscr{C} \to \mathbf{Sets}$ a functor. Then we redefine the *end* of F as $\mathrm{End}(F) := E = \mathrm{Hom}(F,F)$ where E is the equalizer in Proposition 1.2.6.

Theorem 1.2.8 (Tannaka Reconstruction Theorem for **G-Sets** [Tan]). *Let G be a finite group. Let* **G-Sets** *be the category of finite G-sets and let $U\colon$* **G-Sets** $\to$ **Sets** *be the forgetful functor. Then G can be reconstructed as the natural transformations $U \to U$ together with the composition, that is, $G \cong \mathrm{End}(U)$ as groups. In particular, the class of natural transformations $U \to U$ is a finite set.*

Proof. By Lemma 1.1.5, we know that $U \cong \mathrm{Hom}_{\textbf{G-Sets}}(G,-)$ as functors **G-Sets** $\to$ **Sets**. Thus, we have the following canonical isomorphisms as sets, that is, bijective maps:

$$\begin{aligned}\mathrm{End}(U) &= \mathrm{Hom}(U,U)\\ &\cong \mathrm{Hom}(\mathrm{Hom}_{\textbf{G-Sets}}(G,-),\mathrm{Hom}_{\textbf{G-Sets}}(G,-)) &(1.19)\\ &\cong \mathrm{Hom}_{\textbf{G-Sets}}(G,G) &(1.20)\\ &\cong U(G) &(1.21)\end{aligned}$$

where the bijection in (1.19) is the claim of Lemma 1.2.3, the bijection in (1.20) is the claim of the Yoneda lemma and the bijection in (1.21) is the map α'_G in Lemma 1.1.5. Now, we regard hom-sets as monoids with respect to the composition of morphisms, and $U(G)$ as a group with its former group structure G. We show that each of the bijections is an isomorphism or anti-isomorphism of monoids. Since the chain of bijections ends with a group, by Remark 1.2.1 this is sufficient to prove the claim.

By Lemma 1.2.3, we know that the bijection in (1.19) is an isomorphism of monoids. The bijection in (1.20) is the map $f \mapsto f^*$ in the Yoneda lemma, which is an anti-isomorphism of monoids because $(f_1 \circ f_2)^* = f_2^* \circ f_1^*$. We show that $f = \alpha'_G$ is an isomorphism of monoids: Let $\varphi_1, \varphi_2 \in \mathrm{Hom}_{\textbf{G-Sets}}(G,G)$. Then we have

$$f(\varphi_1 \circ \varphi_2) = U(\varphi_1 \circ \varphi_2)(1) = U(\varphi_1)(U(\varphi_2)(1)) = U(\varphi_1)(1) \cdot U(\varphi_2)(1) = f(\varphi_1) \cdot f(\varphi_2)$$

by the same arguments as in Remark 1.1.27.

Summing up, we get $\mathrm{End}(U) \cong G$ as groups. □

In our setting we do not know the group G and thus do not have the category **G-Sets**. We only have a category $\mathscr{C}$ and a functor $F\colon \mathscr{C} \to$ **Sets** such that there exists an equivalence of categories $Z\colon \mathscr{C} \to$ **G-Sets** for some group G with $F \cong U \circ Z$. Yet, we still want to reconstruct G. To this end, we show that we can also apply the theorem above for $\mathscr{C}$ and F instead of **G-Sets** and U. For this, we first need a lemma.

Lemma 1.2.9. *Let $\mathscr{C}, \mathscr{D}$ be categories and $F, F'\colon \mathscr{C} \to \mathscr{D}$ be functors. Then any natural transformation $\alpha\colon F \to F'$ is already determined by its values on representatives of isomorphism classes of objects in $\mathscr{C}$. Additionally, let $\mathscr{C}'$ be a full subcategory of $\mathscr{C}$ such that* $\mathrm{Obj}(\mathscr{C}')$ *contains at least one representative of each isomorphism class of objects in $\mathscr{C}$. Then any natural transformation $\beta\colon F|_{\mathscr{C}'} \to F'|_{\mathscr{C}'}$ extends uniquely to a natural transformation $F \to F'$.*

Proof. Let $\alpha\colon F \to F'$ be a natural transformation, let C_1 and C_2 be objects in $\mathscr{C}$ and let $\varphi\colon C_1 \to C_2$ be an isomorphism. Then we get the following commutative diagram:

$$\begin{array}{ccc} F(C_1) & \xrightarrow{F(\varphi)} & F(C_2) \\ \downarrow{\scriptstyle \alpha_{C_1}} & & \downarrow{\scriptstyle \alpha_{C_2}} \\ F'(C_1) & \xrightarrow{F'(\varphi)} & F'(C_2) \end{array}$$

In particular, the morphisms $F(\varphi)$ and $F'(\varphi)$ are also isomorphisms and thus we have

$$\alpha_{C_1} = (F'(\varphi))^{-1} \circ \alpha_{C_2} \circ F(\varphi),$$

so α_{C_1} is uniquely determined by α_{C_2}.

Now let $\beta\colon F|_{\mathscr{C}'} \to F'|_{\mathscr{C}'}$ be a natural transformation. Let C be an object in $\mathscr{C}$. By assumption there exists an object C' in $\mathscr{C}'$ such that $C \cong C'$. Fix an isomorphism $\varphi_C\colon C \to C'$. Then we define $\alpha_C := (F'(\varphi_C))^{-1} \circ \beta_{C'} \circ F(\varphi_C)$ and claim that this yields a natural transformation $\alpha\colon F \to F'$. For this, let C_1 and C_2 be objects in $\mathscr{C}$ and $\varphi\colon C_1 \to C_2$ a morphism. We have to show that the following diagram commutes:

$$\begin{array}{ccc} F(C_1) & \xrightarrow{F(\varphi)} & F(C_2) \\ \downarrow{\scriptstyle \alpha_{C_1}} & & \downarrow{\scriptstyle \alpha_{C_2}} \\ F'(C_1) & \xrightarrow{F'(\varphi)} & F'(C_2) \end{array}$$

Every square in the following diagram commutes:

$$\begin{array}{ccccccc} F(C_1) & \xrightarrow{F(\varphi_{C_1})} & F(C_1') & \xrightarrow{F(\varphi_{C_2} \circ \varphi \circ \varphi_{C_1}^{-1})} & F(C_2') & \xrightarrow{F(\varphi_{C_2}^{-1})} & F(C_2) \\ \downarrow{\scriptstyle \alpha_{C_1}} & & \downarrow{\scriptstyle \beta_{C_1'}} & & \downarrow{\scriptstyle \beta_{C_2'}} & & \downarrow{\scriptstyle \alpha_{C_2}} \\ F'(C_1) & \xrightarrow{F'(\varphi_{C_1})} & F'(C_1') & \xrightarrow{F'(\varphi_{C_2} \circ \varphi \circ \varphi_{C_1}^{-1})} & F'(C_2') & \xrightarrow{F'(\varphi_{C_2}^{-1})} & F'(C_2) \end{array}$$

The left and the right square commute by definition of α and because functors commute with taking inverses. The central square commutes by assumption. Note that we need that $\mathscr{C}'$ is a full subcategory since otherwise $\varphi_{C_2} \circ \varphi \circ \varphi_{C_1}^{-1}$ might not be a morphism in $\mathscr{C}'$. Thus, the outer rectangle commutes, which is just the diagram above.

The uniqueness of α follows from the first part.

Additionally, the uniqueness shows that the construction of α is independent of the choice of the isomorphisms φ_C. In particular, for $C' \in \mathrm{Obj}(\mathscr{C}')$ we have $\alpha_{C'} = \beta_{C'}$ because we can choose $\varphi_{C'} = \mathrm{id}_{C'}$. □

Proposition 1.2.10. *Let $\mathscr{C}'$, $\mathscr{C}$, and $\mathscr{D}$ be categories. Let $Z\colon \mathscr{C}' \to \mathscr{D}$ be an equivalence of categories and let $F'\colon \mathscr{C}' \to \mathscr{D}$ and $F\colon \mathscr{C} \to \mathscr{D}$ be functors such that $F' \cong F \circ Z$. That is, the following diagram commutes up to isomorphism of functors:*

$$\begin{array}{ccc} \mathscr{C}' & \xrightarrow{Z} & \mathscr{C} \\ & {\scriptstyle F'} \searrow \quad \swarrow {\scriptstyle F} & \\ & \mathscr{D} & \end{array}$$

Then $\mathrm{Hom}(F', F')$ *and* $\mathrm{Hom}(F, F)$ *are isomorphic as monoids.*

Proof. By Lemma 1.2.3, we have $\mathrm{Hom}(F', F') \cong \mathrm{Hom}(F \circ Z, F \circ Z)$ as monoids. Thus, we only have to show $\mathrm{Hom}(F \circ Z, F \circ Z) \cong \mathrm{Hom}(F, F)$ as monoids. For this, let $\tilde{\mathscr{C}}$ be the subcategory of $\mathscr{C}$ consisting of all objects $Z(C')$ with $C' \in \mathrm{Obj}(\mathscr{C}')$ and all morphisms $Z(\varphi)$ where φ is a morphism in $\mathscr{C}'$. Let $\beta \in \mathrm{Hom}(F \circ Z, F \circ Z)$. Then, for all objects C_1' and C_2' in $\mathscr{C}'$ and all morphisms $\varphi\colon C_1' \to C_2'$ the following diagram commutes:

$$\begin{array}{ccc} F(Z(C_1')) & \xrightarrow{F(Z(\varphi))} & F(Z(C_2')) \\ \downarrow{\scriptstyle \beta_{C_1'}} & & \downarrow{\scriptstyle \beta_{C_2'}} \\ F(Z(C_1')) & \xrightarrow{F(Z(\varphi))} & F(Z(C_2')) \end{array}$$

This implies that β is a natural transformation $F|_{\tilde{\mathscr{C}}} \to F|_{\tilde{\mathscr{C}}}$. Since Z is an equivalence of categories, it is essentially surjective and fully faithful, so by Lemma 1.2.9 we can extend β uniquely to a natural transformation $\alpha\colon F \to F$. This extension is injective since restricting α to $\tilde{\mathscr{C}}$ gives back β. Again by Lemma 1.2.9, every natural transformation $F \to F$ arises in this way. Thus, we have a bijection. It remains to show that this is an isomorphism of monoids. For this, let $\beta_1, \beta_2 \in \mathrm{Hom}(F|_{\tilde{\mathscr{C}}}, F|_{\tilde{\mathscr{C}}})$ and $\beta = \beta_1 \circ \beta_2$. Let α_1, α_2 and α be the extensions of β_1, β_2 and β to $\mathrm{Hom}(F,F)$ respectively. We have to show that $\alpha = \alpha_1 \circ \alpha_2$, that is, that for all objects C in $\mathscr{C}$ we have $\alpha_C = (\alpha_1)_C \circ (\alpha_2)_C$. Let C' be an object in $\tilde{\mathscr{C}}$ with $C \cong C'$ and let $\varphi\colon C \to C'$ be an isomorphism. Then by definition we have

$$\begin{aligned}
\alpha_C &= (F(\varphi))^{-1} \circ \beta_{C'} \circ F(\varphi) \\
&= (F(\varphi))^{-1} \circ (\beta_1)_{C'} \circ (\beta_2)_{C'} \circ F(\varphi) \\
&= (F(\varphi))^{-1} \circ (\beta_1)_{C'} \circ F(\varphi) \circ (F(\varphi))^{-1} \circ (\beta_2)_{C'} \circ F(\varphi) \\
&= (\alpha_1)_C \circ (\alpha_2)_C.
\end{aligned}$$

This finishes the proof. □

Corollary 1.2.11. *Let $\mathscr{C}$ be a category equivalent to* **G-Sets** *via an equivalence $Z\colon \mathscr{C} \to$* **G-Sets** *and let $F\colon \mathscr{C} \to$* **Sets** *be a functor such that $F \cong U \circ Z$. Then* $\mathrm{End}(F) \cong G$ *as groups.*

Proof. By Remark 1.2.7, Proposition 1.2.10, and Theorem 1.2.8, we have

$$\mathrm{End}(F) = \mathrm{Hom}(F,F) \cong \mathrm{Hom}(U,U) \cong G$$

as monoids and, since G is a group, also as groups. □

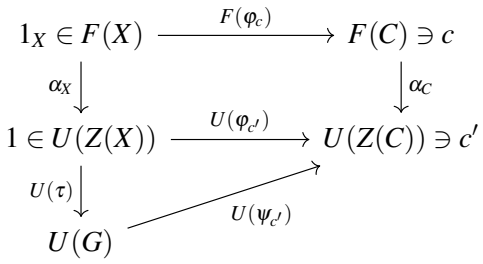


Figure 1.5: The objects and morphisms used in the proof of Lemma 1.3.2

1.3 Reconstructing G algorithmically

From now on, let $\mathscr{C}$ be a category equivalent to **G-Sets** via an equivalence $Z\colon \mathscr{C} \to$ **G-Sets**, let $F\colon \mathscr{C} \to$ **Sets** be a functor such that $F \cong U \circ Z$ and let X be an object in $\mathscr{C}$ with $Z(X) \cong G$. In examples and for testing the implementation of our algorithms we choose $\mathscr{C} =$ **Skeletal-G-Sets**, $F = U \circ R$ where R is the realization functor **Skeletal-G-Sets** $\to$ **G-Sets** and $X = A(G)$ where A is the abstraction functor **G-Sets** $\to$ **Skeletal-G-Sets**. The aim of this chapter is to reconstruct G from $\mathscr{C}$ algorithmically using three different approaches.

1.3.1 $\mathrm{End}(F)$ as an equalizer

Remark 1.3.1. As we have seen in the last chapter, we can reconstruct G from the category $\mathscr{C}$ as an equalizer of two parallel maps $\rho, \lambda : S \to T$. The first problem when it comes to actually computing the equalizer are the *infinite* index classes $\mathrm{Obj}(\mathscr{C})$ of the products which form S and T respectively. If we can find a subcategory $\tilde{I}$ of $\mathscr{C}$ with finitely many objects such that $\mathrm{End}(F|_{\tilde{I}}) \cong \mathrm{End}(F)$ as monoids, then we can compute the equalizer with finite index sets $\mathrm{Obj}(\tilde{I})$. If we think of ends as sets of natural transformations, then computing $\mathrm{End}(F|_{\tilde{I}})$ corresponds to finding only finitely many components of a natural transformation $F \to F$ and to only ensuring that some of the squares in the definition of a natural transformation $F \to F$ commute. The condition $\mathrm{End}(F|_{\tilde{I}}) \cong \mathrm{End}(F)$ means that we can still regain all natural transformations $F \to F$ by uniquely extending natural transformations $F|_{\tilde{I}} \to F|_{\tilde{I}}$. For example for $\mathscr{C} =$ **Skeletal-G-Sets** and $F = U \circ R$ we can choose $\tilde{I}$ as the full subcategory of $\mathscr{C}$ with $\mathrm{Obj}(\tilde{I}) = \{A(G)\}$. We show this in general in Proposition 1.3.4 but first prove a lemma as a preparation.

Lemma 1.3.2. *Let $\alpha\colon F \to U \circ Z$ be an isomorphism and $\tau\colon Z(X) \to G$ a G-equivariant bijection. Set $1_X := \alpha_X^{-1}(\tau^{-1}(1)) \in F(X)$. Then for any object C in $\mathscr{C}$ and any $c \in F(C)$ there exists a unique morphism $\varphi_c\colon X \to C$ with $F(\varphi_c)(1_X) = c$. Additionally, for any object D in $\mathscr{C}$ and any morphism $\varphi\colon C \to D$ we have $\varphi \circ \varphi_c = \varphi_{F(\varphi)(c)}$.*

Proof. We visualize all objects and morphisms used in the proof in Figure 1.5. Let C be an object in $\mathscr{C}$ and $c \in F(C)$. Set $c' := \alpha_C(c) \in U(Z(C))$. Then there exists a G-equivariant map $\psi_{c'}\colon G \to Z(C)$ which maps 1 to c':

$$\begin{aligned} \psi_{c'}\colon G &\to Z(C) \\ g &\mapsto c'g \end{aligned}$$

Thus, we get the G-equivariant map $\varphi_{c'} := \psi_{c'} \circ \tau\colon Z(X) \to Z(C)$ which maps $\tau^{-1}(1) = \alpha_X(1_X)$ to c'. Since $Z(X) \cong G$, we have $\alpha_X(1_X)G = Z(X)$, so any G-equivariant map with source $Z(X)$ is

uniquely determined by the image of the element $\alpha_X(1_X)$. Thus, $\varphi_{c'}$ is the unique G-equivariant map $Z(X) \to Z(C)$ which maps $\alpha_X(1_X)$ to c'. Since Z is an equivalence of categories, there exists a unique morphism $\varphi_c \colon X \to C$ with $Z(\varphi_c) = \varphi_{c'}$. We have the following commutative diagram:

$$\begin{array}{ccc} F(X) & \xrightarrow{F(\varphi_c)} & F(C) \\ \downarrow{\scriptstyle \alpha_X} & & \downarrow{\scriptstyle \alpha_C} \\ U(Z(X)) & \xrightarrow{U(Z(\varphi_c))} & U(Z(C)) \end{array}$$

By construction, we have $F(\varphi_c)(1_X) = \alpha_C^{-1}(U(Z(\varphi_c))(\alpha_X(1_X))) = \alpha_C^{-1}(c') = c$.

We now show the uniqueness of φ_c. Let $\varphi \colon X \to C$ be a morphism with $F(\varphi)(1_X) = c$. By the commutative diagram above, $Z(\varphi)$ is a G-equivariant map which maps $\alpha_X(1_X)$ to c'. By uniqueness of $\varphi_{c'}$ we get $Z(\varphi) = \varphi_{c'}$ and because Z is fully faithful we have $\varphi = \varphi_c$.

Finally, let D be an object in $\mathscr{C}$ and $\varphi \colon C \to D$ a morphism. Then we have

$$F(\varphi \circ \varphi_c)(1_X) = F(\varphi)(F(\varphi_c)(1_X)) = F(\varphi)(c) = F(\varphi_{F(\varphi)(c)})(1_X)$$

and the uniqueness of $\varphi_{F(\varphi)(c)}$ implies $\varphi \circ \varphi_c = \varphi_{F(\varphi)(c)}$. □

Definition 1.3.3. Let I be a set of objects in $\mathscr{C}$. Then we define $\tilde{I}$ to be the full subcategory of $\mathscr{C}$ with $\mathrm{Obj}(\tilde{I}) = I$.

Proposition 1.3.4. *Set* $I := \{X\}$. *Then* $\mathrm{Hom}(F|_{\tilde{I}}, F|_{\tilde{I}}) \cong \mathrm{Hom}(F, F)$ *as monoids.*

Proof. For the proof, fix an isomorphism $\alpha \colon F \to U \circ Z$ as well as a G-equivariant bijection $\tau \colon Z(X) \to G$, and set $1_X := \alpha_X^{-1}(\tau^{-1}(1)) \in F(X)$.

We first show that we have $\mathrm{Hom}(F|_{\tilde{I}}, F|_{\tilde{I}}) \cong \mathrm{Hom}(F, F)$ as sets. Let $(\beta_C)_C \in \mathrm{Hom}(F|_{\tilde{I}}, F|_{\tilde{I}})$. Then $(\beta_C)_C$ only has a single component $\beta_X =: \beta$. We want to extend β to a natural transformation $\alpha \colon F \to F$. For this, let C be an object in $\mathscr{C}$. We want to define the component $\alpha_C \colon F(C) \to F(C)$ of α. For any $c \in F(C)$ we set $\alpha_C(c) := F(\varphi_c)(\beta(1_X))$ where φ_c is the unique morphism from Lemma 1.3.2 with $F(\varphi_c)(1_X) = c$. We visualize this in the following diagram:

$$\begin{array}{ccc} F(X) & \xrightarrow{F(\varphi_c)} & F(C) \\ \downarrow{\scriptstyle \beta} & & \downarrow{\scriptstyle \alpha_C} \\ F(X) & \xrightarrow{F(\varphi_c)} & F(C) \end{array}$$

Note that we do not make a claim about the commutativity of the diagram. We have to check that α is indeed a natural transformation $F \to F$. Let C_1 and C_2 be two objects in $\mathscr{C}$ and $\varphi \colon C_1 \to C_2$ a morphism. We have to show that the following diagram commutes:

$$\begin{array}{ccc} F(C_1) & \xrightarrow{F(\varphi)} & F(C_2) \\ \downarrow{\scriptstyle \alpha_{C_1}} & & \downarrow{\scriptstyle \alpha_{C_2}} \\ F(C_1) & \xrightarrow{F(\varphi)} & F(C_2) \end{array}$$

Let $c \in C_1$. Then we have

$$\begin{aligned}(F(\varphi) \circ \alpha_{C_1})(c) &= F(\varphi)(F(\varphi_c)(\beta(1_X))) \\ &= F(\varphi \circ \varphi_c)(\beta(1_X)) \\ &= F(\varphi_{F(\varphi)(c)})(\beta(1_X)) \\ &= (\alpha_{C_2} \circ F(\varphi))(c)\end{aligned}$$

by Lemma 1.3.2, that is, $F(\varphi) \circ \alpha_{C_1} = \alpha_{C_2} \circ F(\varphi)$.

We claim that α really is an extension of β, that is, we have $\alpha_X = \beta$. Let $x \in F(X)$. Then by assumption, the following diagram commutes:

$$\begin{array}{ccc} F(X) & \xrightarrow{F(\varphi_x)} & F(X) \\ \downarrow{\scriptstyle \beta} & & \downarrow{\scriptstyle \beta} \\ F(X) & \xrightarrow{F(\varphi_x)} & F(X) \end{array}$$

Thus, we have $\alpha_X(x) = F(\varphi_x)(\beta(1_X)) = \beta(F(\varphi_x)(1_X)) = \beta(x)$.

Finally, we have to show that any natural transformation $\alpha\colon F \to F$ arises in this way, that is, if we choose $\beta := \alpha_X$ and extend β as above, we get back α. Thus, we have to show $\alpha_C(c) = F(\varphi_c)(\alpha_X(1_X))$ for any object C in $\mathscr{C}$ and any $c \in F(C)$. By assumption, the following diagram commutes:

$$\begin{array}{ccc} F(X) & \xrightarrow{F(\varphi_c)} & F(C) \\ \downarrow{\scriptstyle \alpha_X} & & \downarrow{\scriptstyle \alpha_C} \\ F(X) & \xrightarrow{F(\varphi_c)} & F(C) \end{array}$$

Thus, we get $F(\varphi_c)(\alpha_X(1_X)) = \alpha_C(F(\varphi_c)(1_X)) = \alpha_C(c)$.

Summing up, we have a bijection $\mathrm{Hom}(F|_{\tilde{I}}, F|_{\tilde{I}}) \cong \mathrm{Hom}(F,F)$. It remains to show that this is an isomorphism of monoids. Let $\beta_1, \beta_2 \in \mathrm{Hom}(F|_{\tilde{I}}, F|_{\tilde{I}})$ and set $\beta := \beta_1 \circ \beta_2$. Let α_1, α_2 and α be the extensions of β_1, β_2 and β respectively. We have to show $\alpha = \alpha_1 \circ \alpha_2$. As a preparation we show $F(\varphi_{\beta_2(1_X)})(\beta_1(1_X)) = \beta_1(F(\varphi_{\beta_2(1_X)})(1_X))$. This follows immediately from the following diagram, which commutes by assumption:

$$\begin{array}{ccc} F(X) & \xrightarrow{F(\varphi_{\beta_2(1_X)})} & F(C) \\ \downarrow{\scriptstyle \beta_1} & & \downarrow{\scriptstyle \beta_1} \\ F(X) & \xrightarrow{F(\varphi_{\beta_2(1_X)})} & F(C) \end{array}$$

Now, let C be an object in $\mathscr{C}$ and $c \in C$. Then we have

$$\begin{aligned}(\alpha_1)_C((\alpha_2)_C(c)) &= (\alpha_1)_C(F(\varphi_c)(\beta_2(1_X))) \\ &= F(\varphi_{F(\varphi_c)(\beta_2(1_X))})(\beta_1(1_X)) \\ &= F(\varphi_c)(F(\varphi_{\beta_2(1_X)})(\beta_1(1_X))) \\ &= F(\varphi_c)(\beta_1(F(\varphi_{\beta_2(1_X)})(1_X))) \\ &= F(\varphi_c)(\beta_1(\beta_2(1_X))) \\ &= F(\varphi_c)(\beta(1_X)) \\ &= \alpha_C(c).\end{aligned}$$

This finishes the proof. □

Remark 1.3.5 (Implementation)**.** Summing up, we have shown that $\mathrm{End}(F|_{\tilde{I}}) \cong G$ with $I := \{X\}$. For the implementation we need some remarks.

First, note that U is faithful because morphisms in **G-Sets** are G-equivariant maps and two morphism are equal if and only if they are equal as maps. In particular, any hom-set $\mathrm{Hom}_{\textbf{G-Sets}}(\Omega_1,\Omega_2)$ in **G-Sets** is finite because $\mathrm{Hom}_{\textbf{Sets}}(U(\Omega_1),U(\Omega_2))$ has $|U(\Omega_2)|^{|U(\Omega_1)|} < \infty$ elements. Since Z is fully faithful, this implies that all hom-sets in $\mathscr{C}$ are finite. Also note that we can easily compute $\mathrm{Hom}_{\textbf{Sets}}(M,N)$ for finite sets M and N as $\prod_{m\in M} N$.

If we have implementations of $\mathscr{C}$ and F in CAP and the implementation of a method computing hom-sets in $\mathscr{C}$, we can now implement an algorithm which reconstructs G by computing $\mathrm{End}(F|_{\tilde{I}})$ as the equalizer of two parallel maps $\rho,\lambda\colon S \to T$ as in Proposition 1.2.6: The index sets of the products forming S and T are finite, so we only have to compute the finitely many components of ρ and λ. These components are maps between finite sets formed by taking finite hom-sets in $\mathscr{C}$ and in **Sets**. Thus, we can compute all sets involved explicitly. Using the implementation of `FinSets` in [BM18], we can also represent and compose maps between finite sets. Since ρ and λ are defined via taking F and composing maps, we have all algorithms we need. However, the sets get too large: even for $G = S_3$ and $\mathscr{C} =$ **Skeletal-G-Sets** we already have $|T| = (6^6)^6 \approx 1.03 \cdot 10^{28}$. The solution is to work in `SkeletalFinSets` instead, which is also implemented in [BM18]. That is, we only store the size n of a set and think of the set as $\{1,\dots,n\}$. However, we want to compose elements of hom-sets. Thus, for $M = \{1,\dots,m\}$ and $N = \{1,\dots,n\}$ we need a mapping between integers in $\{1,\dots,n^m\} \cong \mathrm{Hom}_{\textbf{Sets}}(M,N)$ and actual maps $M \to N$. For this, we set $\tilde{M} := \{0,\dots,m-1\}$ and $\tilde{N} := \{0,\dots,n-1\}$. Let $f\colon \tilde{M} \to \tilde{N}$ be a map. We can think of f as the vector $(f(m))_{m\in M}$ with $0 \le f(m) < n-1$ for all $m \in M$. This is exactly the base-n representation with m digits of some integer in $\{0,\dots,n^m-1\}$. Conversely, any base-n representation of an integer in $\{0,\dots,n^m-1\}$ padded to m digits is a map $f\colon \tilde{M} \to \tilde{N}$ in this way. By subtracting 1 before the conversion and adding 1 back after the conversion, we can use this to identify the integers in $\{1,\dots,n^m\}$ with maps $M \to N$.

Now we have all algorithms for the implementation in `EndAsEqualizer`. The implementation computes the equalizer in Proposition 1.2.6 for an arbitrary finite index set. The functions for converting back and forth between integers and morphisms are called `IntToMorphism` and `MorphismToInt`. Additionally, the function `PseudoMorphismToInt` allows us to omit wrapping a morphism in a GAP object in case we would immediately unwrap it again using `MorphismToInt`. The functions `GetRhoComponent` and `GetLambdaComponent` compute ρ_{C_1,C_2} and λ_{C_1,C_2} for two objects C_1 and C_2 as given in Proposition 1.2.6. The main function constructs S, T, ρ and λ using the functions above and the implementation of the direct product in `SkeletalFinSets`, computes the equalizer of ρ and λ using the implementation in `SkeletalFinSets` and finally translates the resulting skeletal set into a set of morphisms again.

Remark 1.3.6 (Search space)**.** In Remark 1.3.5 we use a skeletal representation of sets to avoid having to construct S and T as sets explicitly. However, for computing the components of ρ and λ we still have to construct the elements of S explicitly. For $I = \{X\}$ this means that we have to construct all maps in $\mathrm{Hom}_{\textbf{Sets}}(F(X),F(X))$. We define the *search space* of an algorithm as the subset Y of $\mathrm{Hom}_{\textbf{Sets}}(F(X),F(X))$ consisting of the maps which we construct explicitly during the algorithm. Of course, for a valid result the search space must contain the equalizer of ρ and λ. If this is the case and Y is a proper subset of $\mathrm{Hom}_{\textbf{Sets}}(F(X),F(X))$, we only have to check if $\rho(y) = \lambda(y)$ for the elements $y \in Y$ instead of checking this for all elements in $\mathrm{Hom}_{\textbf{Sets}}(F(X),F(X))$.

In Remark 1.3.5 the search space is the whole set $\mathrm{Hom}_{\textbf{Sets}}(F(X),F(X))$, which has size $|F(X)|^{|F(X)|}$.

For $X = G = S_3$ we have to construct $6^6 = 46656$ maps, which is still feasible. However, for a group of order 100 we have no chance to construct all 100^{100} maps as we cannot even to loop over their representing integers in feasible time. Thus, we have to find strategies which reduce the size of the search space. A first step is to only consider bijective maps in $\mathrm{Hom}_{\mathbf{Sets}}(F(X),F(X))$. This works because we have shown that the natural transformations $F \to F$ form a group and thus their components have to be bijective maps because inverses exist. However, this still leaves us with $|F(X)|!$ elements. One further step would be to try to find a different index set I leading to a smaller set S. For example, for $G = S_3$ and $\mathscr{C} =$ **Skeletal-G-Sets** a natural idea would be to choose $I := \{A(U_2\backslash S_3), A(U_3\backslash S_3)\}$ because $A(U_2\backslash S_3) \times A(U_3\backslash S_3) = A(\{1\}\backslash G)$. However, $\mathrm{End}(F|_I)$ contains 54 elements instead of 6. The reason for this is that $\mathrm{Hom}_{\mathbf{G\text{-}Sets}}(C_2\backslash S_3, C_2\backslash S_3)$ only contains a single element, that is, the identity, although we would expect three elements by mapping $\bar{1} \in C_2\backslash S_3$ to any element of $C_2\backslash S_3$. However, the two non-identity maps are not well-defined. On the other hand, $\mathrm{Hom}_{\mathbf{G\text{-}Sets}}(A_3\backslash S_3, A_3\backslash S_3)$ contains two element as expected. It follows that $\mathrm{Hom}_{\mathbf{G\text{-}Sets}}(C_2\backslash S_3, C_2\backslash S_3) \times \mathrm{Hom}_{\mathbf{G\text{-}Sets}}(A_3\backslash S_3, A_3\backslash S_3)$ contains two elements although $\mathrm{Hom}_{\mathbf{G\text{-}Sets}}(C_2\backslash S_3 \times A_3\backslash S_3, C_2\backslash S_3 \times A_3\backslash S_3)$ contains six elements. So we simply do not get enough conditions on the components of our natural transformation. All other choices of I either lead to a larger set S or do not contain more information than I. Thus, we will take a different approach in the next section.

1.3.2 Lifting components of natural maps

Recall the setting which we have fixed above: $\mathscr{C}$ is a category equivalent to **G-Sets** via an equivalence $Z\colon \mathscr{C} \to$ **G-Sets**, $F\colon \mathscr{C} \to$ **Sets** is a functor such that $F \cong U \circ Z$ and X is an object in $\mathscr{C}$ with $Z(X) \cong G$.

Now, we present an algorithm which recursively uses information of maps $F(X') \to F(X')$ for objects X' with $|F(X')| < |F(X)|$ to find maps $F(X) \to F(X)$ which are components of natural transformations $F \to F$. To illustrate the idea behind the algorithm, we start with an example.

Example 1.3.7. Let $\mathscr{C} =$ **Skeletal-G-Sets**, $F = U \circ R$, $X = G$ and $I = \{X\}$. We want to determine $\mathrm{End}(F|_I)$. We know that any element $\beta \in \mathrm{End}(F|_I)$ extends uniquely to a natural transformation $\alpha\colon F \to F$. Then for all G-equivariant maps $\varphi_1\colon \{1\}\backslash S_3 \to A_3\backslash S_3$ and $\varphi_2\colon A_3\backslash S_3 \to S_3\backslash S_3$ we have the following commutative diagram:

$$\begin{array}{ccccc} \{1\}\backslash S_3 & \xrightarrow{\varphi_1} & A_3\backslash S_3 & \xrightarrow{\varphi_2} & S_3\backslash S_3 \\ \downarrow{\scriptstyle \alpha_{\{1\}\backslash S_3}} & & \downarrow{\scriptstyle \alpha_{A_3\backslash S_3}} & & \downarrow{\scriptstyle \alpha_{S_3\backslash S_3}} \\ \{1\}\backslash S_3 & \xrightarrow{\varphi_1} & A_3\backslash S_3 & \xrightarrow{\varphi_2} & S_3\backslash S_3 \end{array}$$

Since there exists only one map $S_3\backslash S_3 \to S_3\backslash S_3$, we know the component $\alpha_{S_3\backslash S_3}$. The right square imposes no conditions on $\alpha_{A_3\backslash S_3}$, so we can only use the knowledge that $\alpha_{A_3\backslash S_3}$ is bijective. Since $|A_3\backslash S_3| = 2$, this leaves two options for $\alpha_{A_3\backslash S_3}$: the identity on $A_3\backslash S_3$ and the map that swaps the two elements of $A_3\backslash S_3$. We first choose $\alpha_{A_3\backslash S_3} = \mathrm{id}_{A_3\backslash S_3}$. Then the left square imposes conditions on $\alpha_{\{1\}\backslash S_3}$: Let $x \in \{1\}\backslash S_3$. By the commutativity of the left square we must have

$$\alpha_{\{1\}\backslash S_3}(x) \in \varphi_1^{-1}(\alpha_{A_3\backslash S_3}(\varphi_1(x)))$$

for any G-equivariant map $\varphi_1\colon \{1\}\backslash S_3 \to A_3\backslash S_3$, that is,

$$\alpha_{\{1\}\backslash S_3}(x) \in \bigcap_{\varphi \in \mathrm{Hom}_{\mathbf{G\text{-}Sets}}(\{1\}\backslash S_3, A_3\backslash S_3)} \varphi^{-1}(\alpha_{A_3\backslash S_3}(\varphi(x))).$$

One can check that this gives three options for $\alpha_{\{1\}\backslash S_3}(x)$, which we call *lifts* of $\alpha_{A_3\backslash S_3}$. Since $|\{1\}\backslash S_3| = 6$, this gives $3^6 = 729$ possibilities for $\alpha_{\{1\}\backslash S_3}$. If we take the bijectivity into account we can reduce this number to 36. For the second choice of $\alpha_{A_3\backslash S_3}$ we again get 36 possibilities for $\alpha_{\{1\}\backslash S_3}$. Thus, we only have to check if $\rho(\alpha) = \lambda(\alpha)$ are equal for $36+36 = 72$ maps α, that is, our search space has size 72 instead of $6! = 720$.

To get an idea of the size of the search space in general, in the following lemma we compute the size of a single preimage occurring in the algorithm.

Lemma 1.3.8. *Let $\Omega = U\backslash G$ and $\Delta = V\backslash G$, and let $\varphi\colon \Omega \to \Delta$ be a G-equivariant map. Then for any $\bar{y} \in \Delta$ it holds $|\varphi^{-1}(\bar{y})| = \frac{|U\backslash G|}{|V\backslash G|}$.*

Proof. Set $Vg := \varphi(\bar{1})$. Then we have

$$\begin{aligned}
|\varphi^{-1}(y)| &= |\{\bar{x} \in U\backslash G\colon \varphi(\bar{x}) = \bar{y}\}| \\
&= \frac{|\{x \in G\colon Vgx = Vy\}|}{|U|} \\
&= \frac{|\{x \in G\colon \exists v \in V\colon gx = vy\}|}{|U|} \\
&= \frac{|\{g^{-1}vy\colon v \in V\}|}{|U|} \\
&= \frac{|V|}{|U|} \\
&= \frac{|U\backslash G|}{|V\backslash G|}.
\end{aligned}$$

□

Example 1.3.9. Let G be a group of order 9 and $U \in \mathscr{U}$ be a subgroup of order 3. Let $\mathscr{C} =$ **Skeletal-G-Sets**, $F = U \circ R$, $X = G$, $I = \{X\}$ and $\alpha\colon F \to F$ by a natural transformation. Set $\Omega := \{1\}\backslash G$ and $\Delta := U\backslash G$. Then there exist 3^3 maps $\Delta \to \Delta$ and thus, there are 27 possibilities for $\alpha_\Delta\colon \Delta \to \Delta$. We know that there exists a morphism $\varphi\colon \Omega \to \Delta$, for example by applying Lemma 1.3.2. If we fix one possibility of $\alpha_\Delta\colon \Delta \to \Delta$, then as in Example 1.3.7 we get that for any $x \in \Omega$ we have $\alpha_\Omega(x) \in \varphi^{-1}(\alpha_\Delta(\varphi(x)))$. Applying Lemma 1.3.8 gives that $|\varphi^{-1}(\alpha_\Delta(\varphi(x)))| = \frac{9}{3} = 3$. Thus, we have $3^9 = 19683$ possibilities for α_Ω. Taking the 27 possibilities of $\alpha_\Delta\colon \Delta \to \Delta$ into account, we get a total of $19683 \cdot 27 = 531441$ possibilities. Checking 531441 maps is a matter of minutes, whereas checking all $9^9 = 387420489$ maps $\Omega \to \Omega$ would be a matter of hours. Note that there exist three morphisms $\Omega \to \Delta$, so by taking the intersection as in Example 1.3.7, we might reduce the number of possibilities even more.

Remark 1.3.10. The ideas from the examples above are still not applicable to groups of, for example, order 100. Let G be a group of order 100 with a subgroup $U \in \mathscr{U}$ of order 2. Then analogously to the last example, we get $2^{100} \approx 1.27 \cdot 10^{30}$ possible lifts of a single map $\alpha_{U\backslash G}\colon U\backslash G \to U\backslash G$. Thus, we have to restrict the possibilities

$$\alpha_{\{1\}\backslash G}(x) \in \bigcap_{\varphi \in \mathrm{Hom}_{\textbf{G-Sets}}(\{1\}\backslash G, U\backslash G)} \varphi^{-1}(\alpha_{U\backslash G}(\varphi(x)))$$

even more. For this, let $V \in \mathscr{U}$ be another subgroup of G. Then we have

$$\alpha_{\{1\}\backslash G}(x) \in \bigcap_{\varphi \in \mathrm{Hom}_{\mathbf{G\text{-}Sets}}(\{1\}\backslash G, U\backslash G)} \varphi^{-1}(\alpha_{U\backslash G}(\varphi(x))) \cap \bigcap_{\varphi \in \mathrm{Hom}_{\mathbf{G\text{-}Sets}}(\{1\}\backslash G, V\backslash G)} \varphi^{-1}(\alpha_{V\backslash G}(\varphi(x))),$$

that is, we do not lift single maps, but pairs of maps. Finally, this is sufficient for many groups up to order 200. For maximal efficiency, we choose V in such a way that there exists no morphism $U\backslash G \to V\backslash G$ and vice versa, since otherwise, some of the morphisms $\{1\}\backslash G \to V\backslash G$ are simply concatenations $\{1\}\backslash G \to U\backslash G \to V\backslash G$ (or vice versa) and do not add new restrictions. Summing up, this gives Algorithm 10 and Algorithm 11.

Algorithm 10: `LiftMaps`

Input: objects C, C_1 and C_2 in $\mathscr{C}$, a set M_1 of bijective maps $F(C_1) \to F(C_1)$ and a set M_2 of bijective maps $F(C_2) \to F(C_2)$

Output: the set M of bijective maps $f\colon F(C) \to F(C)$ with the following property: there exist $f_1 \in M_1$ and $f_2 \in M_2$ such that for all $\varphi \in \mathrm{Hom}_{\mathscr{C}}(C, C_1)$ we have $F(\varphi) \circ f = f_1 \circ F(\varphi)$ and for all $\varphi \in \mathrm{Hom}_{\mathscr{C}}(C, C_2)$ we have $F(\varphi) \circ f = f_2 \circ F(\varphi)$

```
let M be the empty set;
forall f_1 ∈ M_1 do
    forall f_2 ∈ M_2 do
        forall x ∈ F(C) do
            set P_{x,1} := ⋂_{φ∈Hom_𝒞(C,C_1)} (F(φ))^{-1}(f_1(F(φ)(x)));
            set P_{x,2} := ⋂_{φ∈Hom_𝒞(C,C_2)} (F(φ))^{-1}(f_2(F(φ)(x)));
            set P_x := P_{x,1} ∩ P_{x,2};
        let M' be the set of bijective maps f: F(C) → F(C) with f(x) ∈ P_x for all x ∈ F(C);
        set M := M ∪ M';
return M;
```

Proposition 1.3.11. *Algorithm 10 terminates and yields the correct result.*

Proof. Since for all objects C in $\mathscr{C}$ we have that $F(C) \cong U(Z(C))$ is a finite set, all loops loop over finite sets and the algorithm terminates. Let $f\colon F(C) \to F(C)$ be a bijective map such that there exist $f_1 \in M_1$ and $f_2 \in M_2$ such that for all $\varphi \in \mathrm{Hom}_{\mathscr{C}}(C, C_1)$ we have $F(\varphi) \circ f = f_1 \circ F(\varphi)$ and for all $\varphi \in \mathrm{Hom}_{\mathscr{C}}(C, C_2)$ we have $F(\varphi) \circ f = f_2 \circ F(\varphi)$. Then for all $x \in F(C)$, for all $\varphi_1 \in \mathrm{Hom}_{\mathscr{C}}(C, C_1)$ and for all $\varphi_2 \in \mathrm{Hom}_{\mathscr{C}}(C, C_2)$ we have $f(x) \in (F(\varphi_1))^{-1}(f_1(F(\varphi_1)(x)))$ and $f(x) \in (F(\varphi_2))^{-1}(f_2(F(\varphi_2)(x)))$, that is, $f(x) \in P_x$ in the iteration $f_1 \in M_1$ and $f_2 \in M_2$ and f gets constructed in this iteration. Conversely, let f be constructed in the iteration $f_1 \in M$ and $f_2 \in M$. Then for all $\varphi \in \mathrm{Hom}_{\mathscr{C}}(C, C_1)$ we have $F(\varphi) \circ f = f_1 \circ F(\varphi)$ and for all $\varphi \in \mathrm{Hom}_{\mathscr{C}}(C, C_2)$ we have $F(\varphi) \circ f = f_2 \circ F(\varphi)$ by construction. □

Proposition 1.3.12. *Algorithm 11 terminates and yields the correct result. Additionally, the set M_{C_1} contains exactly the maps $C \to C$ which are components of elements in* $\mathrm{End}\, F|_{\tilde{I}}$ *for $I = \{X\}$.*

Proof. We show that the algorithm terminates and yields the correct result by induction on t':

For $t' = s$ we have $|F(C)| = 1$. Thus, the only map $F(C) \to F(C)$ is the identity and any natural transformation $\alpha\colon F \to F$ has to fulfill $\alpha_C = \mathrm{id}_{F(C)}$. We return $\{\mathrm{id}_{F(C)}\}$ and the algorithm terminates.

Algorithm 11: `LiftEfficiently`

Input: objects C_t in $\mathscr{C}$ for $t \in \{1,\dots,s\}$ such that $C_1 = X$, $|F(C_s))| = 1$ and $|F(C_{t_1})| \geq |F(C_{t_2})|$ if $t_1 \leq t_2$ and a fixed object $C = C_{t'}$

Output: a set M_C of maps $F(C) \to F(C)$ such that for any natural transformation $\alpha\colon F \to F$ we have $\alpha_C \in M_C$

1 **if** $t' = s$ **then**
2 **return** the set $\{\mathrm{id}_{F(C)}\}$;
3 set $t_1 := \min\{t \mid t > t' \text{ and } \mathrm{Hom}_{\mathscr{C}}(C, C_t) \neq \varnothing\}$;
4 set $t_2 := \min(\{t \mid t > t_1 \text{ and } \mathrm{Hom}_{\mathscr{C}}(C, C_t) \neq \varnothing \text{ and } \mathrm{Hom}_{\mathscr{C}}(C_{t_1}, C_t) = \varnothing\} \cup \{s\})$;
5 let M_1 be the result of this algorithm applied to the input C_t for $t \in \{1,\dots,s\}$ and $C = C_{t_1}$;
6 let M_2 be the result of this algorithm applied to the input C_t for $t \in \{1,\dots,s\}$ and $C = C_{t_2}$;
7 let M be the result of the algorithm `LiftMaps` applied to the input C, $C_1 = C_{t_1}$, M_1, $C_2 = C_{t_2}$ and M_2;
8 remove all elements $f \in M$ from M which do not fulfill the following property: for all $\varphi \in \mathrm{Hom}_{\mathscr{C}}(C, C)$ it holds $f \circ F(\varphi) = F(\varphi) \circ f$;
9 **return** M;

Now let $t' < s$ and assume that the algorithm terminates and yields the correct result for all $t \in \{t'+1,\dots,s\}$. We have $|U(Z(C_s))| = |F(C_s)| = 1$. Thus, $Z(C_s)$ has to be the trivial G-set and the unique map $U(A(C)) \to U(A(C_s))$ is G-equivariant, that is, there exists a morphism $A(C) \to A(C_s)$. Since A is fully faithful, there also exists a morphism $C \to C_s$. Therefore, the set $\{C_t \mid t > t' \text{ and } \mathrm{Hom}_{\mathscr{C}}(C, C_t) \neq \varnothing\}$ is non-empty and we find $t_1 \in \{t'+1,\dots,s\}$. Trivially, we find $t_2 \in \{t_1+1,\dots,s\} \subseteq \{t'+1,\dots,s\}$. By induction assumption and Proposition 1.3.11, all other computations in the algorithm terminate. Let $\alpha\colon F \to F$ be a natural transformation. By induction assumption, we have $\alpha_{C_{t_1}} \in M_1$ and $\alpha_{C_{t_2}} \in M_2$. Thus, α_C fulfills the specification of the output of Algorithm 10 by choosing $f_1 = \alpha_{C_{t_1}}$ and $f_2 = \alpha_{C_{t_2}}$. Therefore, we have $\alpha_C \in M$. Additionally, α_C fulfills the condition in line 8 and thus is not removed from M.

It remains to show the additional claim. By Proposition 1.3.4, the components of elements in $\mathrm{End}F|_{\tilde{I}}$ are exactly the components α_X of the natural transformations $\alpha\colon F \to F$ and thus elements of M_{C_1} by the specification of the output of the algorithm. Conversely, any element of the output of the algorithm for $t' = 1$ is a component of an element of $\mathrm{End}F|_{\tilde{I}}$ by line 8. This finishes the proof. □

Remark 1.3.13 (Implementation)**.** The implementation is given in `EndByLifts`. The functions `PermutationsListKWithRestrictions` and `PermutationsListWithRestrictions` are adapted from the GAP functions `PermutationsListK` and `PermutationsList`: they compute all permutations of a given set (this corresponds to *bijective* maps in Algorithm 10) with additional restrictions on the elements at a given position (this corresponds to the condition $f(x) \in P_x$ in Algorithm 10). The functions `LiftMaps` and `LiftEfficiently` are the implementations of Algorithm 10 and Algorithm 11 respectively with some obvious performance optimizations. The main function simply calls `LiftEfficiently` and wraps each element of the result in a tuple to get $\mathrm{End}_{\tilde{I}}(F)$ for $I := \{X\}$.

Remark 1.3.14. Taking preimages and defining maps elementwise only works in **Sets** respectively in categories in which the morphisms are basically maps of sets. Thus, it might be difficult to generalize the algorithm to different underlying categories V. Nevertheless, the basic idea of the algorithm, that is,

lifting the components of a natural transformation from "smaller" to "larger" objects, might also be applicable in more general categories.

Remark 1.3.15. As we have indicated above, lifting maps is feasible for many groups up to order 200. However, there are two main criteria which exclude some groups:

If a group has a prime divisor greater or equal to 11, at some point of the algorithm we have to lift maps $F(\Delta) \to F(\Delta)$ to maps $F(\Omega) \to F(\Omega)$ with $\frac{F(\Omega)}{F(\Delta)} \geq 11$. By Lemma 1.3.8, we possibly get more than $11^{11} = 285311670611$ lifts for a single map $\Delta \to \Delta$. This is not feasible.

If a group is too "well-structured", for example, if it is cyclic, then lifting pairs might not give more conditions than lifting single maps. We have already seen in Remark 1.3.10 that lifting single maps is not feasible in many cases.

Remark 1.3.16. Another idea would be to lift components of a natural transformation $F \to F$ using products. For example, choose $G = S_3$, $\mathscr{C} =$ **Skeletal-G-Sets** and $F = U \circ R$, and let $\alpha\colon F \to F$ be a natural transformation. One can check that $A(U_2\backslash S_3) \times A(U_3\backslash S_3) = A(\{1\}\backslash S_3)$. Thus, we would like to reconstruct the component $\alpha_{A(\{1\}\backslash S_3)}$ using the components $\alpha_{A(U_2\backslash S_3)}$ and $\alpha_{A(U_3\backslash S_3)}$. At first glance this seems easy because in a category with products, the following diagram lifts a morphism $f_{C_1}\colon C_1 \to C_1$ and a morphism $f_{C_2}\colon C_2 \to C_2$ to a unique morphism $f_{C_1} \times f_{C_2}\colon C_1 \times C_2 \to C_1 \times C_2$ by using the universal property of the product:

$$\begin{array}{ccccc} C_1 & \xleftarrow{\pi_{C_1}} & C_1 \times C_2 & \xrightarrow{\pi_{C_2}} & C_2 \\ \downarrow{\scriptstyle f_{C_1}} & & \downarrow{\scriptstyle \exists_1 f_{C_1} \times f_{C_2}} & & \downarrow{\scriptstyle f_{C_2}} \\ C_1 & \xleftarrow{\pi_{C_1}} & C_1 \times C_2 & \xrightarrow{\pi_{C_2}} & C_2 \end{array}$$

In our example, one is tempted to choose $C_1 = A(U_2\backslash S_3)$, $C_2 = A(U_3\backslash S_3)$, $f_{C_1} = \alpha_{A(U_2\backslash S_3)}$ and $f_{C_2} = \alpha_{A(U_3\backslash S_3)}$. Then choosing $\alpha_{C_1 \times C_2}$ for the dashed arrow makes the diagram commute by definition of a natural transformation and because of the uniqueness of the dashed arrow we would conclude $\alpha_{C_1 \times C_2} = f_{C_1} \times f_{C_2}$. However, we have to be more careful since the projections and the components of α are morphisms in different categories. So we try to make this precise:

$$\begin{array}{ccccc} F(C_1) & \xleftarrow{F(\pi_{C_1})} & F(C_1 \times C_2) & \xrightarrow{F(\pi_{C_2})} & F(C_2) \\ \downarrow{\scriptstyle f_{C_1}} & & \downarrow{\scriptstyle \exists_1} & & \downarrow{\scriptstyle f_{C_2}} \\ F(C_1) & \xleftarrow{\pi_{F(C_1)}} & F(C_1) \times F(C_2) & \xrightarrow{\pi_{F(C_2)}} & F(C_2) \\ & {\scriptstyle F(\pi_{C_1})}\nwarrow & & \nearrow{\scriptstyle F(\pi_{C_2})} & \\ & & F(C_1 \times C_2) & & \end{array}$$

Now we see that the we have to apply the universal property of the product $F(C_1) \times F(C_2)$ with regard to projections $\pi_{F(C_1)}$ and $\pi_{F(C_2)}$ in **Sets**, whereas the statement that the diagram commutes was about $F(C_1 \times C_2)$ with regard to $F(\pi_{C_1})$ and $F(\pi_{C_2})$. Thus, for the initial claim we need $\pi_{F(C_1)} = F(\pi_{C_1})$ and $\pi_{F(C_2)} = F(\pi_{C_2})$. However, this is impossible to achieve with our implementation of `SkeletalFinSets` and `FinSets`: the projections $C_1 \times C_2 \to C_1$ and $C_2 \times C_1 \to C_1$ are equal, but the projections $F(C_1) \times F(C_2) \to F(C_1)$ and $F(C_2) \times F(C_1) \to F(C_1)$ are different. This is a

general problem of skeletal categories: by taking isomorphism classes we lose information about the concrete representation of $C_1 \times C_2$ as a product.

1.3.3 Reconstructing the table of marks

Recall the setting which we have fixed above: $\mathscr{C}$ is a category equivalent to **G-Sets** via an equivalence $Z\colon \mathscr{C} \to$ **G-Sets**, $F\colon \mathscr{C} \to$ **Sets** is a functor such that $F \cong U \circ Z$ and X is an object in $\mathscr{C}$ with $Z(X) \cong G$.

A completely different approach to reconstructing the group G is to reconstruct its table of marks T. We can do this even if we do not know the hom-sets. If we know that there exists only one group up to isomorphism with table of marks T, then we have determined G up to isomorphism. However, there are non-isomorphic groups with isomorphic table of marks. For example, the GAP groups `SmallGroup(96,108)` and `SmallGroup(96,114)` have isomorphic table of marks.

Remark 1.3.17 (Idea)**.** The information about the table of marks is encoded in the direct products of **Skeletal-G-Sets**: Let $s,t \in \mathscr{I}$ and $A(U_s\backslash G) \times A(U_t\backslash G) = [m_1,\ldots,m_k]$. Then by Remark 1.1.17 we have the equation $\beta_{U_s}(j)\cdot\beta_{U_t}(j) = \beta_{U_s\backslash G\times U_t\backslash G}(j) = \sum_{i\in\mathscr{I}} m_i\beta_{U_i}(j)$ for all $j \in \mathscr{I}$. If we also take into account that the table of marks is a lower triangular matrix with non-zero diagonal entries, taking all such equations is often sufficient to reconstruct the table of marks.

Remark 1.3.18 (Generalization to $\mathscr{C}$)**.** Since Z is an equivalence of categories and thus commutes with taking products and coproducts, we can generalize the idea in Remark 1.3.17 from **Skeletal-G-Sets** to $\mathscr{C}$: Let L be a minimal set of generators of $\mathscr{C}$ with regard to finite coproducts, that is, any object in $\mathscr{C}$ is isomorphic to a coproduct of elements in L and this decomposition into cofactors is unique up to permutation. Since Z and A are equivalences, the set $L' := \{A(Z(C)) \mid C \in L\}$ has the same properties in **Skeletal-G-Sets**. Then L' must contain the objects $A(U\backslash G)$ with $U \in \mathscr{U}$ because these objects cannot be decomposed into proper cofactors. Conversely, the set $\{A(U\backslash G) \mid U \in \mathscr{U}\}$ already generates **Skeletal-G-Sets** with regard to coproducts. Thus, by minimality of L' we get $L' = \{A(U\backslash G) \mid U \in \mathscr{U}\}$. Therefore, we can apply the idea of Remark 1.3.17 to get equations for the entries of the table of marks by taking products of elements of L and writing the result as a coproduct of elements of L up to isomorphism.

One problem remains: the table of marks is a lower triangular matrix because $|U_i| \le |U_j|$ if $i < j$. However, the elements of L (and thus the elements of L') might be ordered differently or not at all. The condition $|U_i| \le |U_j|$ is equivalent to $|G\backslash U_i| \ge |G\backslash U_j|$. Thus, we want to sort the elements C_t of L in descending order with regard to $|F(C_t)|$. Actually, we can even do this without knowing the functor F:

All orbits in $\{1\}\backslash G \times \{1\}\backslash G$ have length $|G|$. Thus, the transitive cofactors of $\{1\}\backslash G \times \{1\}\backslash G$ have to be isomorphic to $\{1\}\backslash G$, that is, we have $A(\{1\}\backslash G) \times A(\{1\}\backslash G) = m_{\{1\}\backslash G}A(\{1\}\backslash G)$. For any object Ω in L' with $\Omega\times\Omega = m_\Omega\Omega$ we have $m_\Omega = |U(R(\Omega))|$ by considering the sizes of the underlying sets. In particular, $m_{\{1\}\backslash G} = |G|$. Since $|U(R(\Omega))| \le |G|$ for all Ω in L', we can find $A(\{1\}\backslash G)$ among the objects Ω in L' with $\Omega\times\Omega = m_\Omega\Omega$ as the one with the largest multiplicity m_Ω.

If we know the object $A(\{1\}\backslash G)$, we can use that for any object Ω in **Skeletal-G-Sets** all orbits in the product $\{1\}\backslash G \times R(\Omega)$ have length $|G|$. Thus, for any Ω in L' we get the equation $A(\{1\}\backslash G)\times\Omega = n_\Omega A(\{1\}\backslash G)$ with $n_\Omega = |U(R(\Omega))|$ by the same reasoning as above.

Again, because Z is an equivalence and because $F \cong U \circ Z$ we can apply the same algorithm to L and get the values $|F(C_t)|$ for all elements C_t of L'.

Remark 1.3.19 (Implementation)**.** A sample implementation of the algorithm described above is given in `ReconstructTableOfMarks`. The arguments are the category $\mathscr{C}$, the minimal generating set L and a function which decomposes any object in $\mathscr{C}$ into a coproduct with cofactors in L. The function

may return an error if it cannot find unique solutions to the equations. If it finishes without error, all equations have been solved uniquely and thus the resulting matrix T must be the table of marks of G up to a permutation of rows i and j with $|U_i| = |U_j|$ and their corresponding columns. In particular, the group order is just the entry $(1,1)$ and we can check for all groups up to isomorphism of this order if its table of marks is T up to a permutation of rows and columns. If there is only one such group up to isomorphism, we have successfully reconstructed G up to isomorphism.

Chapter 2

The category of skeletal finite G-sets

2.1 Skeletal GAP Categories

2.1.1 IsSkeletalFinGSet (for IsCapCategoryObject and IsCellOfSkeletalCategory)

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

2.1.2 IsSkeletalFinGSetMap (for IsCapCategoryMorphism and IsCellOfSkeletalCategory)

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

2.2 Skeletal Attributes

2.2.1 AsList (for IsSkeletalFinGSet)

▷ `AsList(`*`Omega`*`)` (attribute)
Returns: a GAP set
The GAP set of the list used to construct a finite G-set *`Omega`*, i.e., `AsList( FinGSet( G, `*`L`*` ) ) = `*`L`*.

2.2.2 UnderlyingGroup (for IsSkeletalFinGSet)

▷ `UnderlyingGroup(`*`Omega`*`)` (attribute)
Returns: a group
The group G underlying the finite G-set *`Omega`*.

2.3 Skeletal Constructors

2.3.1 FinGSet (for IsGroup, IsList)

▷ `FinGSet(G, L)` (operation)

Returns: a CAP object

Construct a skeletal finite G-set out of the group `G` and a list `L`, i.e., an object in the CAP category `SkeletalFinGSets`.

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> SetName( S3, "S3" );
gap> w1 := FinGSet( S3, [ 1, 2, 3, 1 ] );
<An object in SkeletalFin-S3-Sets>
gap> IsWellDefined( w1 );
true
gap> w2 := FinGSet( S3, [ 1, 2, 3, 1 ] );
<An object in SkeletalFin-S3-Sets>
gap> IsWellDefined( w2 );
true
gap> w1 = w2;
true
gap> S := FinGSet( S3, [ 1 ] );
<An object in SkeletalFin-S3-Sets>
gap> IsWellDefined( S );
false
gap> S := FinGSet( S3, [ "a", 0, 0, 0 ] );
<An object in SkeletalFin-S3-Sets>
gap> IsWellDefined( S );
false
gap> S := FinGSet( S3, [ -1, 0, 0, 0 ] );
<An object in SkeletalFin-S3-Sets>
gap> IsWellDefined( S );
false
```

2.3.2 MapOfFinGSets (for IsSkeletalFinGSet, IsList, IsSkeletalFinGSet)

▷ `MapOfFinGSets(s, G, t)` (operation)

Returns: a CAP morphism

Construct a map $\phi: s \to t$ of the skeletal finite G-sets `s` and `t`, i.e., a morphism in the CAP category `SkeletalFinGSets`, where `G` is a list of lists describing the graph of ϕ.

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> w1 := FinGSet( S3, [ 1, 2, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> w2 := FinGSet( S3, [ 0, 1, 0, 1 ] );
<An object in SkeletalFinGSets>
gap> imgs1 := [ [ [ 1, (2,3), 2 ] ],
>            [ [ 1, (), 4 ], [ 1, (), 2 ] ],
>            [],
```

```
>            [] ];;
gap> pi1 := MapOfFinGSets( w1, imgs1, w2 );
<A morphism in SkeletalFinGSets>
gap> imgs2 := [ [ [ 1, (), 2 ] ],
>            [ [ 1, (), 4 ], [ 1, (2,3), 2 ] ],
>            [],
>            [] ];;
gap> pi2 := MapOfFinGSets( w1, imgs2, w2 );
<A morphism in SkeletalFinGSets>
gap> pi1 = pi2;
true
gap> # BUT
> imgs1 = imgs2;
false
gap> M := FinGSet( S3, [ 2, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> N := FinGSet( SymmetricGroup( 3 ), [ 2, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 1, (), 2 ], [ 1, (), 2 ] ],
>           [ [ 1, (), 2 ] ],
>           [],
>           [] ];;
gap> phi := MapOfFinGSets( M, imgs, N );
Error, The underlying groups of the source and the range are not the same
  with respect to IsIdenticalObj
gap> phi := MapOfFinGSets( M, [ 1 ], M );
Error, I has the wrong format
gap> phi := MapOfFinGSets( M, [ [ 1 ] ], M );
Error, images must be triples
gap> phi := MapOfFinGSets( M, [ [ [ 1, () ] ] ], M );
Error, images must be triples
gap> phi := MapOfFinGSets( M, [ [ [ 1, (), -1 ] ] ], M );
Error, last entry of an image must be an integer j with 1 <= j <= 4
gap> phi := MapOfFinGSets( M, [ [ [ 1, (), 5 ] ] ], M );
Error, last entry of an image must be an integer j with 1 <= j <= 4
gap> imgs := [ [ [ 1, (), 2 ], [ 1, (), 2 ] ], [ [ 1, (), 2 ] ], [] ];;
gap> phi := MapOfFinGSets( M, imgs, M );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( phi );
false
gap> imgs := [ [ [ 1, (), 2 ] ], [ [ 1, (), 2 ] ], [], [] ];;
gap> phi := MapOfFinGSets( M, imgs, M );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( phi );
false
gap> imgs := [ [ [ 0, (), 2 ], [ 1, (), 2 ] ], [ [ 1, (), 2 ] ], [], [] ];;
gap> phi := MapOfFinGSets( M, imgs, M );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( phi );
false
gap> imgs := [ [ [ 3, (), 2 ], [ 1, (), 2 ] ], [ [ 1, (), 2 ] ], [], [] ];;
gap> phi := MapOfFinGSets( M, imgs, M );
```

```
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( phi );
false
gap> imgs := [ [ [ 1, (), 2 ], [ 1, (), 2 ] ], [ [ 1, "", 2 ] ], [], [] ];;
gap> phi := MapOfFinGSets( M, imgs, M );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( phi );
false
gap> imgs := [ [ [ 1, (), 2 ], [ 1, (), 2 ] ], [ [ 1, (), 3 ] ], [], [] ];;
gap> phi := MapOfFinGSets( M, imgs, M );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( phi );
false
gap> imgs := [ [ [ 1, (), 2 ], [ 1, (), 2 ] ],
>           [ [ 1, (1,2,3), 2 ] ],
>           [],
>           [] ];;
gap> phi := MapOfFinGSets( M, imgs, M );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( phi );
false
```

2.3.3 SkeletalFinGSets (for IsGroup)

▷ SkeletalFinGSets(*G*) (attribute)

Returns: a category

The argument is a group *G*. The output is the category of skeletal finite *G*-Sets.

2.4 Skeletal Examples

2.4.1 Skeletal IsEqualForObjects

Example

```
gap> # Groups have to be the same with respect to IsIdenticalObj
> C6 := CyclicGroup( 6 );
<pc group of size 6 with 2 generators>
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> w1 := FinGSet( C6, [1, 2, 3, 1]);
<An object in SkeletalFinGSets>
gap> IsWellDefined( w1 );
true
gap> w2 := FinGSet( S3, [1, 2, 3, 1]);
<An object in SkeletalFinGSets>
gap> IsWellDefined( w2 );
true
gap> w1 = w2;
Error, the object "An object in SkeletalFinGSets" and the object "An o\
bject in SkeletalFinGSets" do not belong to the same CAP category
```

2.4.2 Skeletal PreCompose

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> S := FinGSet( S3, [ 1, 0, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> R := FinGSet( S3, [ 1, 0, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> T := FinGSet( S3, [ 1, 0, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> psi1 := MapOfFinGSets( S, [ [ [ 1, (1,2), 1 ] ], [], [], [] ], R );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( psi1 );
true
gap> psi2 := MapOfFinGSets( R, [ [ [ 1, (1,2,3), 1 ] ] , [], [], [] ] , T );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( psi2 );
true
gap> PreCompose( psi1, psi2 );
<A morphism in SkeletalFinGSets>
gap> phi := PreCompose( psi1, psi2 );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( phi );
true
gap> Display( phi );
[ [ [ 1, (2,3), 1 ] ], [  ], [  ], [  ] ]
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> S := FinGSet( S3, [ 2, 2, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> R := FinGSet( S3, [ 2, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> T := FinGSet( S3,  [ 2, 1, 1, 0 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 2, (1,2), 1 ], [ 1, (1,2,3), 2 ] ],
>           [ [ 1, (), 2 ], [ 1, (2,3), 2 ] ],
>           [ ],
>           [ ] ];;
gap> psi1 := MapOfFinGSets( S, imgs, R );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( psi1 );
true
gap> imgs := [ [ [ 1, (1,3), 1 ], [ 1, (1,2,3), 3 ] ],
>           [ [ 1, (), 2 ] ],
>           [ ],
>           [ ] ];;
gap> psi2 := MapOfFinGSets( R, imgs, T );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( psi2 );
true
gap> pi := PreCompose( psi1, psi2 );
<A morphism in SkeletalFinGSets>
gap> Display( pi );
```

```
[ [ [ 1, (2,3), 3 ], [ 1, (1,2,3), 2 ] ],
  [ [ 1, (), 2 ], [ 1, (2,3), 2 ] ],
  [  ],
  [  ] ]
gap> G := SymmetricGroup( 0 );;
gap> m := FinGSet( G, [ 3 ] );
<An object in SkeletalFinGSets>
gap> n := FinGSet( G, [ 5 ] );
<An object in SkeletalFinGSets>
gap> p := FinGSet( G, [ 7 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 2, (), 1 ], [ 5, (), 1 ], [ 3, (), 1 ] ] ];;
gap> psi := MapOfFinGSets( m, imgs, n );
<A morphism in SkeletalFinGSets>
gap> imgs := [ [ [ 1, (), 1 ],
>              [ 4, (), 1 ],
>              [ 6, (), 1 ],
>              [ 6, (), 1 ],
>              [ 3, (), 1 ] ] ];;
gap> phi := MapOfFinGSets( n, imgs, p );
<A morphism in SkeletalFinGSets>
gap> alpha := PreCompose( psi, phi );
<A morphism in SkeletalFinGSets>
gap> Display( alpha );
[ [ [ 4, (), 1 ], [ 3, (), 1 ], [ 6, (), 1 ] ] ]
```

2.4.3 Skeletal IdentityMorphism

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> M := FinGSet( S3, [ 1, 2, 1, 2 ] );
<An object in SkeletalFinGSets>
gap> iota := IdentityMorphism( M );
<An identity morphism in SkeletalFinGSets>
gap> IsWellDefined( iota );
true
gap> IsEpimorphism( iota );
true
gap> Display( iota );
[ [ [ 1, (), 1 ] ],
  [ [ 1, (), 2 ], [ 2, (), 2 ] ],
  [ [ 1, (), 3 ] ],
  [ [ 1, (), 4 ], [ 2, (), 4 ] ] ]
```

2.4.4 Skeletal Initial and Terminal Objects

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> S := FinGSet( S3, [ 2, 2, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> u := UniversalMorphismFromInitialObject( S );
```

```
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( u );
true
gap> S := FinGSet( S3, [ 2, 2, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> u := UniversalMorphismIntoTerminalObject( S );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( u );
true
gap> G := SymmetricGroup( 0 );;
gap> m := FinGSet( G, [ 8 ] );
<An object in SkeletalFinGSets>
gap> i := InitialObject( m );
<An object in SkeletalFinGSets>
gap> iota := UniversalMorphismFromInitialObject( m );
<A morphism in SkeletalFinGSets>
gap> AsList( i );
[ 0 ]
gap> t := TerminalObject( m );
<An object in SkeletalFinGSets>
gap> AsList( t );
[ 1 ]
gap> pi := UniversalMorphismIntoTerminalObject( m );
<A morphism in SkeletalFinGSets>
gap> IsIdenticalObj( Range( pi ), t );
true
gap> pi_t := UniversalMorphismIntoTerminalObject( m );
<A morphism in SkeletalFinGSets>
gap> Display( pi_t );
[ [ [ 1, (), 1 ], [ 1, (), 1 ], [ 1, (), 1 ], [ 1, (), 1 ],
      [ 1, (), 1 ], [ 1, (), 1 ], [ 1, (), 1 ], [ 1, (), 1 ] ] ]
gap> pi = pi_t;
true
```

2.4.5 Skeletal LiftAlongMonomorphism

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> M := FinGSet( S3, [ 1, 0, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> N := FinGSet( S3, [ 2, 0, 2, 0 ] );
<An object in SkeletalFinGSets>
gap> O := FinGSet( S3, [ 2, 0, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> tau := MapOfFinGSets( M, [ [ [ 1, (1,2), 1 ] ], [], [], [] ], N );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( tau );
true
gap> imgs := [ [ [ 2, (), 1 ], [ 1, (1,3,2), 1 ] ], [], [], [] ];;
gap> iota := MapOfFinGSets( O, imgs, N );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( iota );
```

```
true
gap> IsMonomorphism( iota );
true
gap> u := LiftAlongMonomorphism( iota, tau );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( u );
true
gap> tau = PreCompose( u, iota );
true
```

2.4.6 Skeletal ColiftAlongEpimorphism

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> M := FinGSet( S3, [ 2, 0, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> N := FinGSet( S3, [ 1, 0, 1, 0 ] );
<An object in SkeletalFinGSets>
gap> O := FinGSet( S3, [ 2, 0, 1, 0 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 1, (1,2), 1 ], [ 1, (), 3 ] ], [], [], [] ];;
gap> tau := MapOfFinGSets( M, imgs, O );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( tau );
true
gap> imgs := [ [ [ 1, (1,2,3), 1 ], [ 1, (1,2), 3 ] ], [], [], [] ];;
gap> epsilon := MapOfFinGSets( M, imgs, N );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( epsilon );
true
gap> IsEpimorphism( epsilon );
true
gap> u := ColiftAlongEpimorphism( epsilon, tau );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( u );
true
gap> tau = PreCompose( epsilon, u );
true
```

2.4.7 Skeletal Product

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> A := FinGSet( S3, [ 0, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> B := FinGSet( S3, [ 0, 0, 1, 0 ] );
<An object in SkeletalFinGSets>
gap> Display( DirectProduct( A, B ) );
[ SymmetricGroup( [ 1 .. 3 ] ), [ 1, 0, 0, 0 ] ]
gap> S4 := SymmetricGroup( 4 );
Sym( [ 1 .. 4 ] )
```

```
gap> A := FinGSet( S4, [ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> B := FinGSet( S4, [ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> Display( DirectProduct( A, B ) );
[ SymmetricGroup( [ 1 .. 4 ] ), [ 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ] ]
gap> pi := ProjectionInFactorOfDirectProduct( [ A, A ], 1 );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( pi );
true
gap> Display( pi );
[ [ [ 1, (), 3 ], [ 1, (), 3 ], [ 1, (), 3 ], [ 1, (), 3 ],
      [ 1, (), 3 ] ], [  ], [ [ 1, (), 3 ], [ 1, (), 3 ] ], [  ],
  [  ], [  ], [  ], [  ], [  ], [  ], [  ] ]
gap> S5 := SymmetricGroup( 5 );
Sym( [ 1 .. 5 ] )
gap> A := FinGSet( S5,
>               [ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
>             );
<An object in SkeletalFinGSets>
gap> B := FinGSet( S5,
>               [ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
>             );
<An object in SkeletalFinGSets>
gap> D := [ A, B ];
[ <An object in SkeletalFinGSets>,
  <An object in SkeletalFinGSets> ]
gap> pi1 := ProjectionInFactorOfDirectProduct( D, 1 );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( pi1 );
true
gap> pi2 := ProjectionInFactorOfDirectProduct( D, 2 );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( pi2 );
true
gap> tau := [ pi1, pi2 ];
[ <A morphism in SkeletalFinGSets>, <A morphism in SkeletalFinGSets> ]
gap> u := UniversalMorphismIntoDirectProduct( D, tau );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( u );
true
gap> A := FinGSet( S3, [ 0, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> B := FinGSet( S3, [ 0, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> pi1 := ProjectionInFactorOfDirectProduct( [ A, B ], 1 );
<A morphism in SkeletalFinGSets>
gap> pi2 := ProjectionInFactorOfDirectProduct( [ A, B ], 2 );
<A morphism in SkeletalFinGSets>
gap> Display( pi1 );
[ [ [ 1, (), 2 ] ], [ [ 1, (), 2 ] ], [  ], [  ] ]
gap> Display( pi2 );
```

```
[ [ [ 1, (1,3), 2 ] ], [ [ 1, (), 2 ] ], [  ], [  ] ]
gap> M := FinGSet( S3, [ 1, 2, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> N := FinGSet( S3, [ 1, 0, 1, 2 ] );
<An object in SkeletalFinGSets>
gap> D := [ M, N ];
[ <An object in SkeletalFinGSets>,
  <An object in SkeletalFinGSets> ]
gap> tau1 := ProjectionInFactorOfDirectProduct( D, 1 );
<A morphism in SkeletalFinGSets>
gap> tau2 := ProjectionInFactorOfDirectProduct( D, 2 );
<A morphism in SkeletalFinGSets>
gap> tau := [ tau1, tau2 ];
[ <A morphism in SkeletalFinGSets>, <A morphism in SkeletalFinGSets> ]
gap> u := UniversalMorphismIntoDirectProduct( D, tau );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( u );
true
gap> Display( u );
[ [ [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 5, (), 1 ], [ 6, (), 1 ], [ 7, (), 1 ], [ 8, (), 1 ],
      [ 9, (), 1 ], [ 10, (), 1 ], [ 11, (), 1 ], [ 12, (), 1 ],
      [ 13, (), 1 ], [ 14, (), 1 ], [ 15, (), 1 ], [ 16, (), 1 ],
      [ 17, (), 1 ], [ 18, (), 1 ] ],
  [ [ 1, (), 2 ], [ 2, (), 2 ], [ 3, (), 2 ], [ 4, (), 2 ] ], [  ],
  [  ] ]
gap> L := FinGSet( S3, [ 2, 1, 0, 1 ] );
<An object in SkeletalFinGSets>
gap> D := [ M, N, L ];
[ <An object in SkeletalFinGSets>,
  <An object in SkeletalFinGSets>,
  <An object in SkeletalFinGSets> ]
gap> tau := ProjectionInFactorOfDirectProduct( D, 3 );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( tau );
true
gap> G := SymmetricGroup( 0 );;
gap> m := FinGSet( G, [ 7 ] );
<An object in SkeletalFinGSets>
gap> n := FinGSet( G, [ 3 ] );
<An object in SkeletalFinGSets>
gap> p := FinGSet( G, [ 4 ] );
<An object in SkeletalFinGSets>
gap> d := DirectProduct( [ m, n, p ] );
<An object in SkeletalFinGSets>
gap> AsList( d );
[ 84 ]
gap> pi1 := ProjectionInFactorOfDirectProduct( [ m, n, p ], 1 );
<A morphism in SkeletalFinGSets>
gap> Display( pi1 );
[ [ [ 1, (), 1 ], [ 1, (), 1 ], [ 1, (), 1 ], [ 1, (), 1 ],
      [ 1, (), 1 ], [ 1, (), 1 ], [ 1, (), 1 ], [ 1, (), 1 ],
```

```
      [ 1, (), 1 ], [ 1, (), 1 ], [ 1, (), 1 ], [ 1, (), 1 ],
      [ 2, (), 1 ], [ 2, (), 1 ], [ 2, (), 1 ], [ 2, (), 1 ],
      [ 2, (), 1 ], [ 2, (), 1 ], [ 2, (), 1 ], [ 2, (), 1 ],
      [ 2, (), 1 ], [ 2, (), 1 ], [ 2, (), 1 ], [ 2, (), 1 ],
      [ 3, (), 1 ], [ 3, (), 1 ], [ 3, (), 1 ], [ 3, (), 1 ],
      [ 3, (), 1 ], [ 3, (), 1 ], [ 3, (), 1 ], [ 3, (), 1 ],
      [ 3, (), 1 ], [ 3, (), 1 ], [ 3, (), 1 ], [ 3, (), 1 ],
      [ 4, (), 1 ], [ 4, (), 1 ], [ 4, (), 1 ], [ 4, (), 1 ],
      [ 4, (), 1 ], [ 4, (), 1 ], [ 4, (), 1 ], [ 4, (), 1 ],
      [ 4, (), 1 ], [ 4, (), 1 ], [ 4, (), 1 ], [ 4, (), 1 ],
      [ 5, (), 1 ], [ 5, (), 1 ], [ 5, (), 1 ], [ 5, (), 1 ],
      [ 5, (), 1 ], [ 5, (), 1 ], [ 5, (), 1 ], [ 5, (), 1 ],
      [ 5, (), 1 ], [ 5, (), 1 ], [ 5, (), 1 ], [ 5, (), 1 ],
      [ 6, (), 1 ], [ 6, (), 1 ], [ 6, (), 1 ], [ 6, (), 1 ],
      [ 6, (), 1 ], [ 6, (), 1 ], [ 6, (), 1 ], [ 6, (), 1 ],
      [ 6, (), 1 ], [ 6, (), 1 ], [ 6, (), 1 ], [ 6, (), 1 ],
      [ 7, (), 1 ], [ 7, (), 1 ], [ 7, (), 1 ], [ 7, (), 1 ],
      [ 7, (), 1 ], [ 7, (), 1 ], [ 7, (), 1 ], [ 7, (), 1 ],
      [ 7, (), 1 ], [ 7, (), 1 ], [ 7, (), 1 ], [ 7, (), 1 ] ] ]
gap> pi3 := ProjectionInFactorOfDirectProduct( [ m, n, p ], 3 );
<A morphism in SkeletalFinGSets>
gap> Display( pi3 );
[ [ [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ],
      [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ] ] ]
```

2.4.8 Skeletal Coproduct

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> M1 := FinGSet( S3, [ 2, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> M2 := FinGSet( S3, [ 1, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
```

```
gap> M3 := FinGSet( S3, [ 2, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> M4 := FinGSet( S3, [ 2, 0, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> C := Coproduct( [ M1, M2, M3, M4 ] );
<An object in SkeletalFinGSets>
gap> tau1 := InjectionOfCofactorOfCoproduct( [ M1, M2, M3, M4 ], 1 );
<A morphism in SkeletalFinGSets>
gap> tau2 := InjectionOfCofactorOfCoproduct( [ M1, M2, M3, M4 ], 2 );
<A morphism in SkeletalFinGSets>
gap> tau3 := InjectionOfCofactorOfCoproduct( [ M1, M2, M3, M4 ], 3 );
<A morphism in SkeletalFinGSets>
gap> tau4 := InjectionOfCofactorOfCoproduct( [ M1, M2, M3, M4 ], 4 );
<A morphism in SkeletalFinGSets>
gap> tau := [ tau1, tau2, tau3, tau4 ];;
gap> D := [ M1, M2, M3, M4 ];;
gap> id_to_be := UniversalMorphismFromCoproduct( D, tau );
<A morphism in SkeletalFinGSets>
gap> id := IdentityMorphism( C );
<An identity morphism in SkeletalFinGSets>
gap> id_to_be = id;
true
gap> T := TerminalObject( M1 );
<An object in SkeletalFinGSets>
gap> pi1 := UniversalMorphismIntoTerminalObject( M1 );
<A morphism in SkeletalFinGSets>
gap> pi2 := UniversalMorphismIntoTerminalObject( M2 );
<A morphism in SkeletalFinGSets>
gap> pi3 := UniversalMorphismIntoTerminalObject( M3 );
<A morphism in SkeletalFinGSets>
gap> pi4 := UniversalMorphismIntoTerminalObject( M4 );
<A morphism in SkeletalFinGSets>
gap> pi := [ pi1, pi2, pi3, pi4 ];;
gap> psi := UniversalMorphismFromCoproduct( D, pi );
<A morphism in SkeletalFinGSets>
gap> psi = UniversalMorphismIntoTerminalObject( C );
true
gap> IsEpimorphism( psi );
true
gap> G := SymmetricGroup( 0 );;
gap> m := FinGSet( G, [ 7 ] );
<An object in SkeletalFinGSets>
gap> n := FinGSet( G, [ 3 ] );
<An object in SkeletalFinGSets>
gap> p := FinGSet( G, [ 4 ] );
<An object in SkeletalFinGSets>
gap> c := Coproduct( m, n, p );
<An object in SkeletalFinGSets>
gap> AsList( c );
[ 14 ]
gap> iota1 := InjectionOfCofactorOfCoproduct( [ m, n, p ], 1 );
<A morphism in SkeletalFinGSets>
```

```
gap> IsWellDefined( iota1 );
true
gap> Display( iota1 );
[ [ [ 1, (), 1 ],
    [ 2, (), 1 ],
    [ 3, (), 1 ],
    [ 4, (), 1 ],
    [ 5, (), 1 ],
    [ 6, (), 1 ],
    [ 7, (), 1 ] ] ]
gap> iota3 := InjectionOfCofactorOfCoproduct( [ m, n, p ], 3 );
<A morphism in SkeletalFinGSets>
gap> Display( iota3 );
[ [ [ 11, (), 1 ], [ 12, (), 1 ], [ 13, (), 1 ], [ 14, (), 1 ] ] ]
```

2.4.9 Skeletal Image

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> M := FinGSet( S3, [ 2, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 1, (), 2 ], [ 1, (), 2 ] ], [ [ 1, (), 2 ] ], [], [] ];;
gap> phi := MapOfFinGSets( M, imgs, M );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( phi );
true
gap> IsEpimorphism( phi );
false
gap> I := ImageObject( phi );
<An object in SkeletalFinGSets>
gap> iota := ImageEmbedding( phi );
<A monomorphism in SkeletalFinGSets>
gap> phi_res := CoastrictionToImage( phi );
<A morphism in SkeletalFinGSets>
gap> phi = PreCompose( phi_res, iota );
true
gap> T := FinGSet( S3, [ 1, 1, 0, 0 ] );;
gap> imgs := [ [ [ 1, (), 2 ], [ 1, (), 2 ] ], [ [ 1, (), 2 ] ], [], [] ];;
gap> tau1 := MapOfFinGSets( M, imgs, T );;
gap> imgs := [ [ [ 1, (), 1 ] ], [ [ 1, (), 2 ] ], [], [] ];;
gap> tau2 := MapOfFinGSets( T, imgs, M );;
gap> IsMonomorphism( tau2 );
true
gap> phi = PreCompose( tau1, tau2 );
true
gap> u := UniversalMorphismFromImage( phi, [ tau1, tau2 ] );
<A morphism in SkeletalFinGSets>
gap> tau1 = PreCompose( phi_res, u );
true
gap> iota = PreCompose( u, tau2 );
true
gap> S3 := SymmetricGroup( 3 );
```

```
Sym( [ 1 .. 3 ] )
gap> M := FinGSet( S3, [ 2, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 2, (), 1 ], [ 1, (), 1 ] ], [ [ 1, (), 2 ] ], [], [] ];;
gap> phi := MapOfFinGSets( M, imgs, M );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( phi );
true
gap> IsEpimorphism( phi );
true
gap> I := ImageObject( phi );
<An object in SkeletalFinGSets>
gap> psi := ImageEmbedding( phi );
<A monomorphism in SkeletalFinGSets>
gap> phi_res := CoastrictionToImage( phi );
<A morphism in SkeletalFinGSets>
gap> phi = PreCompose( phi_res, psi );
true
gap> G := SymmetricGroup( 0 );;
gap> m := FinGSet( G, [ 7 ] );
<An object in SkeletalFinGSets>
gap> n := FinGSet( G, [ 3 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 7, (), 1 ], [ 5, (), 1 ], [ 5, (), 1 ] ] ];;
gap> phi := MapOfFinGSets( n, imgs, m );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( phi );
true
gap> ImageObject( phi );
<An object in SkeletalFinGSets>
gap> AsList( ImageObject( phi ) );
[ 2 ]
gap> pi := ImageEmbedding( phi );
<A monomorphism in SkeletalFinGSets>
gap> Display( pi );
[ [ [ 5, (), 1 ], [ 7, (), 1 ] ] ]
gap> phi_res := CoastrictionToImage( phi );
<A morphism in SkeletalFinGSets>
gap> phi = PreCompose( phi_res, pi );
true
```

2.4.10 Skeletal Equalizer

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> s := FinGSet( S3, [ 1, 0, 2, 0 ] );
<An object in SkeletalFinGSets>
gap> r := FinGSet( S3, [ 1, 2, 1, 0 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 1, (1,2), 1 ] ],
>           [],
>           [ [ 1, (), 3 ], [ 1, (1,2,3), 3 ] ],
```

```
>           [] ];;
gap> psi1 := MapOfFinGSets( s, imgs, r );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( psi1 );
true
gap> imgs := [ [[ 1, (1,2), 3 ]], [], [[ 1, (), 3 ], [ 1, (), 3 ]], [] ];;
gap> psi2 := MapOfFinGSets( s, imgs, r );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( psi2 );
true
gap> D := [ psi1, psi2 ];;
gap> Eq := Equalizer( D );
<An object in SkeletalFinGSets>
gap> AsList( Eq );
[ 0, 0, 2, 0 ]
gap> psi := EmbeddingOfEqualizer( D );
<A monomorphism in SkeletalFinGSets>
gap> IsWellDefined( psi );
true
gap> Display( psi );
[ [  ], [  ], [ [ 1, (), 3 ], [ 2, (), 3 ] ], [  ] ]
gap> PreCompose( psi, psi1 ) = PreCompose( psi, psi2 );
true
gap> t := FinGSet( S3, [ 1, 0, 1, 0 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 2, (1,2), 3 ] ], [], [ [ 1, (1,2,3), 3 ] ], [] ];;
gap> tau := MapOfFinGSets( t, imgs , s );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( tau );
true
gap> phi := UniversalMorphismIntoEqualizer( D, tau );
<A morphism in SkeletalFinGSets>
gap> Display( phi );
[ [ [ 2, (1,2), 3 ] ], [  ], [ [ 1, (1,2,3), 3 ] ], [  ] ]
gap> IsWellDefined( phi );
true
gap> PreCompose( phi, psi ) = tau;
true
gap> G := SymmetricGroup( 0 );;
gap> S := FinGSet( G, [ 5 ] );
<An object in SkeletalFinGSets>
gap> T := FinGSet( G, [ 3 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 3, (), 1 ],
>              [ 3, (), 1 ],
>              [ 1, (), 1 ],
>              [ 2, (), 1 ],
>              [ 2, (), 1 ] ] ];;
gap> f1 := MapOfFinGSets( S, imgs, T );
<A morphism in SkeletalFinGSets>
gap> imgs := [ [ [ 3, (), 1 ],
>              [ 2, (), 1 ],
```

```
>             [ 3, (), 1 ],
>             [ 1, (), 1 ],
>             [ 2, (), 1 ] ] ];;
gap> f2 := MapOfFinGSets( S, imgs, T );
<A morphism in SkeletalFinGSets>
gap> imgs := [ [ [ 3, (), 1 ],
>             [ 1, (), 1 ],
>             [ 2, (), 1 ],
>             [ 1, (), 1 ],
>             [ 2, (), 1 ] ] ];;
gap> f3 := MapOfFinGSets( S, imgs, T );
<A morphism in SkeletalFinGSets>
gap> D := [ f1, f2, f3 ];;
gap> Eq := Equalizer( D );
<An object in SkeletalFinGSets>
gap> AsList( Eq );
[ 2 ]
gap> psi := EmbeddingOfEqualizer( D );
<A monomorphism in SkeletalFinGSets>
gap> Display( psi );
[ [ [ 1, (), 1 ], [ 5, (), 1 ] ] ]
gap> PreCompose( psi, f1 ) = PreCompose( psi, f2 );
true
gap> PreCompose( psi, f1 ) = PreCompose( psi, f3 );
true
gap> D := [ f2, f3 ];
[ <A morphism in SkeletalFinGSets>, <A morphism in SkeletalFinGSets> ]
gap> Eq := Equalizer( D );
<An object in SkeletalFinGSets>
gap> AsList( Eq );
[ 3 ]
gap> psi := EmbeddingOfEqualizer( D );
<A monomorphism in SkeletalFinGSets>
gap> Display( psi );
[ [ [ 1, (), 1 ], [ 4, (), 1 ], [ 5, (), 1 ] ] ]
```

2.4.11 Skeletal Pullback

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> A := FinGSet( S3, [ 1, 0, 1, 0 ] );
<An object in SkeletalFinGSets>
gap> B := FinGSet( S3, [ 2, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> C := FinGSet( S3, [ 2, 1, 1, 0 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 2, (), 1 ] ], [], [ [ 1, (), 3 ] ], [] ];;
gap> tau1 := MapOfFinGSets( A, imgs, C );
<A morphism in SkeletalFinGSets>
gap> imgs := [ [ [ 1, (), 1 ], [ 2, (), 1 ] ], [ [ 1, (), 2 ] ], [], [] ];;
gap> tau2 := MapOfFinGSets( B, imgs, C );
<A morphism in SkeletalFinGSets>
```

```
gap> D := [ tau1, tau2 ];
[ <A morphism in SkeletalFinGSets>,
 <A morphism in SkeletalFinGSets> ]
gap> F := FiberProduct( D );
<An object in SkeletalFinGSets>
gap> Display( F );
[ SymmetricGroup( [ 1 .. 3 ] ), [ 1, 0, 0, 0 ] ]
gap> pi1 := ProjectionInFactorOfFiberProduct( D, 1 );
<A morphism in SkeletalFinGSets>
gap> Display( pi1 );
[ [ [ 1, (), 1 ] ], [  ], [  ], [  ] ]
gap> pi2 := ProjectionInFactorOfFiberProduct( D, 2 );
<A morphism in SkeletalFinGSets>
gap> Display( pi2 );
[ [ [ 2, (), 1 ] ], [  ], [  ], [  ] ]
gap> G := SymmetricGroup( 0 );;
gap> m := FinGSet( G, [ 5 ] );
<An object in SkeletalFinGSets>
gap> n1 := FinGSet( G, [ 3 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ] ] ];;
gap> iota1 := MapOfFinGSets( n1, imgs, m );
<A morphism in SkeletalFinGSets>
gap> IsMonomorphism( iota1 );
true
gap> n2 := FinGSet( G, [ 4 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 4, (), 1 ] ] ];;
gap> iota2 := MapOfFinGSets( n2, imgs, m );
<A morphism in SkeletalFinGSets>
gap> IsMonomorphism( iota2 );
true
gap> D := [ iota1, iota2 ];
[ <A monomorphism in SkeletalFinGSets>, <A monomorphism in SkeletalFinGSets> ]
gap> Fib := FiberProduct( D );
<An object in SkeletalFinGSets>
gap> AsList( Fib );
[ 3 ]
gap> pi1 := ProjectionInFactorOfFiberProduct( D, 1 );
<A monomorphism in SkeletalFinGSets>
gap> Display( pi1 );
[ [ [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ] ] ]
gap> int1 := ImageObject( pi1 );
<An object in SkeletalFinGSets>
gap> AsList( int1 );
[ 3 ]
gap> pi2 := ProjectionInFactorOfFiberProduct( D, 2 );
<A monomorphism in SkeletalFinGSets>
gap> Display( pi2 );
[ [ [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ] ] ]
gap> int2 := ImageObject( pi2 );
<An object in SkeletalFinGSets>
```

```
gap> AsList( int2 );
[ 3 ]
gap> omega1 := PreCompose( pi1, iota1 );
<A monomorphism in SkeletalFinGSets>
gap> omega2 := PreCompose( pi2, iota2 );
<A monomorphism in SkeletalFinGSets>
gap> omega1 = omega2;
true
gap> Display( omega1 );
[ [ [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ] ] ]
```

2.4.12 Skeletal Coequalizer

Example

```
gap> S5 := SymmetricGroup( 5 );
Sym( [ 1 .. 5 ] )
gap> g_1_1 := ();;
gap> g_1_2 := (1,2);;
gap> g_1_3 := (1,3);;
gap> g_1_4 := (1,4);;
gap> g_2_1 := ();;
gap> g_2_2 := (1,2);;
gap> g_2_3 := ();;
gap> g_2_4 := (1,2);;
gap> g_3_1 := ();;
gap> g_3_2 := (1,2);;
gap> g_3_3 := (1,3);;
gap> g_3_4 := (1,3);;
gap> A := FinGSet( S5,
>               [ 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
>             );
<An object in SkeletalFinGSets>
gap> B := FinGSet( S5,
>               [ 0, 3, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
>             );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 1, g_1_1, 2 ],
>             [ 1, g_1_3, 2 ],
>             [ 2, g_2_1, 2 ],
>             [ 2, g_2_3, 2 ],
>             [ 3, g_3_1, 2 ],
>             [ 3, g_3_3, 2 ] ],
>           [], [], [], [],
>           [], [], [], [],
>           [], [], [], [],
>           [], [], [], [], [], [] ];;
gap> f_1 := MapOfFinGSets(A, imgs, B);
<A morphism in SkeletalFinGSets>
gap> imgs := [ [ [ 1, g_1_2, 4 ],
>             [ 1, g_1_4, 4 ],
>             [ 2, g_2_2, 4 ],
>             [ 2, g_2_4, 4 ],
>             [ 3, g_3_2, 4 ],
```

```
>             [ 3, g_3_4, 4 ] ],
>           [], [], [], [],
>           [], [], [], [],
>           [], [], [], [],
>           [], [], [], [], [], [] ];;
gap> f_2 := MapOfFinGSets(A, imgs, B);
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( f_1 );
true
gap> IsWellDefined( f_2 );
true
gap> D := [ f_1, f_2 ];;
gap> Cq := Coequalizer( D );
<An object in SkeletalFinGSets>
gap> AsList( Cq );
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1 ]
gap> pi := ProjectionOntoCoequalizer( D );
<An epimorphism in SkeletalFinGSets>
gap> IsWellDefined( pi );
true
gap> id_to_be := UniversalMorphismFromCoequalizer( D, pi );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( id_to_be );
true
gap> id := IdentityMorphism( Cq );
<An identity morphism in SkeletalFinGSets>
gap> id = id_to_be;
true
gap> A := FinGSet( S5,
>               [ 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
>             );
<An object in SkeletalFinGSets>
gap> B := FinGSet( S5,
>               [ 0, 3, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
>             );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 1, g_1_1, 2 ],
>             [ 1, g_1_3, 2 ],
>             [ 2, g_2_1, 2 ],
>             [ 2, g_2_3, 2 ],
>             [ 3, g_3_1, 2 ],
>             [ 3, g_3_3, 2 ] ],
>           [], [], [], [],
>           [], [], [], [],
>           [], [], [], [],
>           [], [], [], [], [], [] ];;
gap> f_1 := MapOfFinGSets(A, imgs, B);
<A morphism in SkeletalFinGSets>
gap> imgs := [ [ [ 1, g_1_2, 4 ],
>             [ 1, g_1_4, 4 ],
>             [ 2, g_2_2, 4 ],
>             [ 2, g_2_4, 4 ],
```

```
>             [ 3, g_3_2, 4 ],
>             [ 3, g_3_4, 4 ] ],
>           [], [], [], [],
>           [], [], [], [],
>           [], [], [], [],
>           [], [], [], [], [], [] ];;
gap> f_2 := MapOfFinGSets(A, imgs, B);
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( f_1 );
true
gap> IsWellDefined( f_2 );
true
gap> D := [ f_1, f_2 ];;
gap> Cq := Coequalizer( D );
<An object in SkeletalFinGSets>
gap> AsList( Cq );
[ 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1 ]
gap> pi := ProjectionOntoCoequalizer( D );
<An epimorphism in SkeletalFinGSets>
gap> IsWellDefined( pi );
true
gap> id_to_be := UniversalMorphismFromCoequalizer( D, pi );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( id_to_be );
true
gap> id := IdentityMorphism( Cq );
<An identity morphism in SkeletalFinGSets>
gap> id = id_to_be;
true
gap> A := FinGSet( S5,
>               [ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
>             );
<An object in SkeletalFinGSets>
gap> B := FinGSet( S5,
>               [ 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
>             );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 1, (), 3 ] ],
>           [], [], [], [],
>           [], [], [], [],
>           [], [], [], [],
>           [], [], [], [], [], [] ];;
gap> f_1 := MapOfFinGSets(A, imgs, B);
<A morphism in SkeletalFinGSets>
gap> imgs := [ [ [ 1, (1,2,3), 4] ],
>           [], [], [], [],
>           [], [], [], [],
>           [], [], [], [],
>           [], [], [], [], [], [] ];;
gap> f_2 := MapOfFinGSets(A, imgs, B);
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( f_1 );
```

```
true
gap> IsWellDefined( f_2 );
true
gap> D := [ f_1, f_2 ];;
gap> Cq := Coequalizer( D );
<An object in SkeletalFinGSets>
gap> AsList( Cq );
[ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 ]
gap> pi := ProjectionOntoCoequalizer( D );
<An epimorphism in SkeletalFinGSets>
gap> IsWellDefined( pi );
true
gap> G := SymmetricGroup( 0 );;
gap> s := FinGSet( G, [ 5 ] );
<An object in SkeletalFinGSets>
gap> t := FinGSet( G, [ 4 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 3, (), 1 ],
>              [ 4, (), 1 ],
>              [ 4, (), 1 ],
>              [ 2, (), 1 ],
>              [ 4, (), 1 ] ] ];;
gap> f := MapOfFinGSets( s, imgs, t );
<A morphism in SkeletalFinGSets>
gap> imgs := [ [ [ 3, (), 1 ],
>              [ 3, (), 1 ],
>              [ 4, (), 1 ],
>              [ 2, (), 1 ],
>              [ 4, (), 1 ] ] ];;
gap> g := MapOfFinGSets( s, imgs, t );
<A morphism in SkeletalFinGSets>
gap> D := [ f, g ];
[ <A morphism in SkeletalFinGSets>, <A morphism in SkeletalFinGSets> ]
gap> C := Coequalizer( D );
<An object in SkeletalFinGSets>
gap> AsList( C );
[ 3 ]
gap> pi := ProjectionOntoCoequalizer( D );
<An epimorphism in SkeletalFinGSets>
gap> Display( pi );
[ [ [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ], [ 3, (), 1 ] ] ]
gap> imgs := [ [ [ 2, (), 1 ], [ 1, (), 1 ], [ 2, (), 1 ], [ 2, (), 1 ] ] ];;
gap> tau := MapOfFinGSets( t, imgs, FinGSet( G, [ 2 ] ) );
<A morphism in SkeletalFinGSets>
gap> phi := UniversalMorphismFromCoequalizer( D, tau );
<A morphism in SkeletalFinGSets>
gap> Display( phi );
[ [ [ 2, (), 1 ], [ 1, (), 1 ], [ 2, (), 1 ] ] ]
gap> PreCompose( pi, phi ) = tau;
true
gap> G := SymmetricGroup( 0 );;
gap> A := FinGSet( G, [ 2 ] );
```

```
<An object in SkeletalFinGSets>
gap> B := FinGSet( G, [ 3 ] );
<An object in SkeletalFinGSets>
gap> f := MapOfFinGSets( A, [ [ [ 1, (), 1 ], [ 2, (), 1 ] ] ], B );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( f );
true
gap> g := MapOfFinGSets( A, [ [ [ 2, (), 1 ], [ 3, (), 1 ] ] ], B );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( g );
true
gap> D := [ f, g ];;
gap> Cq := Coequalizer( D );;
gap> Display( Cq );
[ Group( () ), [ 1 ] ]
gap> pi := ProjectionOntoCoequalizer( D );;
gap> IsWellDefined( pi );
true
gap> PreCompose( f, pi ) = PreCompose( g, pi );
true
gap> G := SymmetricGroup( 3 );;
gap> A := FinGSet( G, [ 0, 0, 0, 0 ] );;
gap> id := IdentityMorphism( A );
<An identity morphism in SkeletalFinGSets>
gap> D := [ id, id ];;
gap> Cq := Coequalizer( D );;
gap> Display( Cq );
[ SymmetricGroup( [ 1 .. 3 ] ), [ 0, 0, 0, 0 ] ]
gap> pi := ProjectionOntoCoequalizer( D );;
gap> IsWellDefined( pi );
true
gap> pi = id;
true
gap> G := SymmetricGroup( 3 );;
gap> A := FinGSet( G, [ 0, 0, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> B := FinGSet( G, [ 1, 1, 1, 1 ] );
<An object in SkeletalFinGSets>
gap> f := MapOfFinGSets( A, [ [ ], [ ], [ ], [ ] ], B );
<A morphism in SkeletalFinGSets>
gap> IsWellDefined( f );
true
gap> D := [ f, f ];;
gap> Cq := Coequalizer( D );;
gap> Display( Cq );
[ SymmetricGroup( [ 1 .. 3 ] ), [ 1, 1, 1, 1 ] ]
gap> pi := ProjectionOntoCoequalizer( D );;
gap> pi = IdentityMorphism( B );
true
```

2.4.13 Skeletal Pushout

Example

```
gap> S3 := SymmetricGroup( 3 );
Sym( [ 1 .. 3 ] )
gap> A := FinGSet( S3, [ 1, 0, 1, 0 ] );
<An object in SkeletalFinGSets>
gap> B := FinGSet( S3, [ 2, 1, 0, 0 ] );
<An object in SkeletalFinGSets>
gap> C := FinGSet( S3, [ 3, 1, 1, 0 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 3, (), 1 ] ], [], [ [ 1, (), 3 ] ], [] ];;
gap> tau1 := MapOfFinGSets( A, imgs, C );
<A morphism in SkeletalFinGSets>
gap> imgs := [ [ [ 2, (), 1 ], [ 3, (), 1 ] ], [ [ 1, (), 2 ] ], [], [] ];;
gap> tau2 := MapOfFinGSets( B, imgs, C );
<A morphism in SkeletalFinGSets>
gap> D := [ tau1, tau2 ];
[ <A morphism in SkeletalFinGSets>,
 <A morphism in SkeletalFinGSets> ]
gap> F := FiberProduct( D );
<An object in SkeletalFinGSets>
gap> pi1 := ProjectionInFactorOfFiberProduct( D, 1 );
<A morphism in SkeletalFinGSets>
gap> pi2 := ProjectionInFactorOfFiberProduct( D, 2 );
<A morphism in SkeletalFinGSets>
gap> iota := UniversalMorphismFromPushout( [ pi1, pi2 ], [ tau1, tau2 ] );
<A morphism in SkeletalFinGSets>
gap> Display( iota );
[ [ [ 3, (), 1 ], [ 2, (), 1 ] ], [ [ 1, (), 2 ] ], [ [ 1, (), 3 ] ], [  ] ]
gap> G := SymmetricGroup( 0 );;
gap> M := FinGSet( G, [ 5 ] );
<An object in SkeletalFinGSets>
gap> N1 := FinGSet( G, [ 3 ] );
<An object in SkeletalFinGSets>
gap> imgs := [ [ [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ] ] ];;
gap> iota1 := MapOfFinGSets( N1, imgs, M );
<A morphism in SkeletalFinGSets>
gap> IsMonomorphism( iota1 );
true
gap> N2 := FinGSet( G, [ 2 ] );
<An object in SkeletalFinGSets>
gap> iota2 := MapOfFinGSets( N2, [ [ [ 1, (), 1 ], [ 2, (), 1 ] ] ], M );
<A morphism in SkeletalFinGSets>
gap> IsMonomorphism( iota2 );
true
gap> D := [ iota1, iota2 ];
[ <A monomorphism in SkeletalFinGSets>, <A monomorphism in SkeletalFinGSets> ]
gap> Fib := FiberProduct( D );
<An object in SkeletalFinGSets>
gap> AsList( Fib );
[ 2 ]
gap> pi1 := ProjectionInFactorOfFiberProduct( D, 1 );
<A monomorphism in SkeletalFinGSets>
```

```
gap> Display( pi1 );
[ [ [ 1, (), 1 ], [ 2, (), 1 ] ] ]
gap> pi2 := ProjectionInFactorOfFiberProduct( D, 2 );
<A monomorphism in SkeletalFinGSets>
gap> Display( pi2 );
[ [ [ 1, (), 1 ], [ 2, (), 1 ] ] ]
gap> D := [ pi1, pi2 ];
[ <A monomorphism in SkeletalFinGSets>, <A monomorphism in SkeletalFinGSets> ]
gap> UU := Pushout( D );
<An object in SkeletalFinGSets>
gap> AsList( UU );
[ 3 ]
gap> kappa1 := InjectionOfCofactorOfPushout( D, 1 );
<A morphism in SkeletalFinGSets>
gap> Display( kappa1 );
[ [ [ 1, (), 1 ], [ 2, (), 1 ], [ 3, (), 1 ] ] ]
gap> kappa2 := InjectionOfCofactorOfPushout( D, 2 );
<A morphism in SkeletalFinGSets>
gap> Display( kappa2 );
[ [ [ 1, (), 1 ], [ 2, (), 1 ] ] ]
gap> PreCompose( pi1, kappa1 ) = PreCompose( pi2, kappa2 );
true
```

Chapter 3

Tools

3.1 Helper functions

3.1.1 PositionAndConjugatorOfStabilizer

▷ `PositionAndConjugatorOfStabilizer(`*`args...`*`)` (function)

Returns: a list with first entry an integer and second entry a group element

Returns a pair `[ i, g ]` such that `ConjugateSubgroup( S, g ) = RepresentativeTom( ToM, i )` where `S = Stabilizer( `*`args...`*` )` and `ToM = TableOfMarks( `*`args`*`[1] )`. See `Stabilizer` for a specification of *`args`*.

Example

```
gap> G := SymmetricGroup( 3 );;
gap> ToM := TableOfMarks( G );;
gap> S := Stabilizer( G, 3 );
Sym( [ 1 .. 2 ] )
gap> pos_and_conj := PositionAndConjugatorOfStabilizer( G, 3 );;
gap> pos := pos_and_conj[1];
2
gap> conj := pos_and_conj[2];
(1,3)
gap> ConjugateSubgroup( S, conj ) = RepresentativeTom( ToM, pos );
true
```

Chapter 4

Reconstructing G from the category of skeletal finite G-sets

4.1 Reconstruction Tools

4.1.1 EndAsEqualizer

▷ `EndAsEqualizer(`*`C, HomC, ForgetfulFunctor, IndexSet`*`)` (function)

See Remark 1.3.5.

4.1.2 EndByLifts

▷ `EndByLifts(`*`C, HomC, ForgetfulFunctor, Objects`*`)` (function)

See Remark 1.3.13.

4.1.3 ReconstructTableOfMarks

▷ `ReconstructTableOfMarks(`*`C, MinimalGeneratingSet, Decompose`*`)` (function)

See Remark 1.3.19.

4.1.4 HomSkeletalFinGSets

▷ `HomSkeletalFinGSets(`*`S, T`*`)` (function)
Returns: a finite set (see `FinSetsForCAP`)
The finite set $\mathrm{Hom}_{\mathrm{SkeletalFinGSets}}(S,T)$.

4.1.5 ForgetfulFunctorSkeletalFinGSets (for IsGroup)

▷ `ForgetfulFunctorSkeletalFinGSets(`*`G`*`)` (attribute)
Returns: a functor SkeletalFinGSets → SkeletalFinSets
The forgetful functor SkeletalFinGSets → SkeletalFinSets.

4.1.6 ReconstructGroup

▷ `ReconstructGroup(`*`C, HomC, ForgetfulFunctor, GeneratingSet, EndImplementation`*`)` (function)

Returns: a group

The input is a CAP category `C` which is equivalent to the category of skeletal finite G-sets for some group G, a function `HomC` computing homs in `C` (e.g. `HomSkeletalFinGSets`), a generating set of `C`, and a function computing ends (e.g. `EndAsEqualizer` or `EndByLifts`). The output is a group isomorphic to G.

4.2 Examples

4.2.1 Reconstructing the Table of Marks

Example

```
gap> G := CyclicGroup( 210 );;
gap> ToM := MatTom( TableOfMarks( G ) );
[ [ 210, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 105, 105, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 70, 0, 70, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 42, 0, 0, 42, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 35, 35, 35, 0, 35, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 30, 0, 0, 0, 0, 30, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 21, 21, 0, 21, 0, 0, 21, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 15, 15, 0, 0, 0, 15, 0, 15, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 14, 0, 14, 14, 0, 0, 0, 0, 14, 0, 0, 0, 0, 0, 0, 0 ],
  [ 10, 0, 10, 0, 0, 10, 0, 0, 0, 10, 0, 0, 0, 0, 0, 0 ],
  [ 7, 7, 7, 7, 7, 0, 7, 0, 7, 0, 7, 0, 0, 0, 0, 0 ],
  [ 6, 0, 0, 6, 0, 6, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0 ],
  [ 5, 5, 5, 0, 5, 5, 0, 5, 0, 5, 0, 0, 5, 0, 0, 0 ],
  [ 3, 3, 0, 3, 0, 3, 3, 3, 0, 0, 0, 3, 0, 3, 0, 0 ],
  [ 2, 0, 2, 2, 0, 2, 0, 0, 2, 2, 0, 2, 0, 0, 2, 0 ],
  [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ] ]
gap> k := Size( ToM );
16
gap> minimal_generating_set := [ ];;
gap> for i in [ 1 .. k ] do
>     M := ListWithIdenticalEntries( k, 0 )
>     ; M[ i ] := 1; Add( minimal_generating_set, FinGSet( G, M ) ); od;
gap> Decompose := function( Omega, minimal_generating_set )
>     return List( [ 1 .. k ], i ->
>         AsList( Omega )[Position( AsList( minimal_generating_set[i] ), 1 )]
>     ); end;;
#@if IsPackageMarkedForLoading( "FinSetsForCAP", ">= 2018.09.17" )
gap> computed_ToM := ReconstructTableOfMarks(
>     SkeletalFinGSets( G ),
>     minimal_generating_set,
>     Decompose
> );;
gap> computed_ToM = ToM;
true
#@fi
```

4.2.2 Reconstructing the Group

Example

```
gap> G_1 := CyclicGroup( 5 );;
gap> G_2 := SmallGroup( 20, 5 );;
#@if IsPackageMarkedForLoading( "FinSetsForCAP", ">= 2018.09.17" )
gap> CapCategorySwitchLogicOff( FinSets );
gap> DeactivateCachingOfCategory( FinSets );
gap> DisableSanityChecks( FinSets );
gap> DeactivateCachingOfCategory( SkeletalFinSets );
gap> CapCategorySwitchLogicOff( SkeletalFinSets );
gap> DisableSanityChecks( SkeletalFinSets );
#@fi
gap> CapCategorySwitchLogicOff( SkeletalFinGSets( G_1 ) );
gap> DeactivateCachingOfCategory( SkeletalFinGSets( G_1 ) );
gap> DisableSanityChecks( SkeletalFinGSets( G_1 ) );
gap> CapCategorySwitchLogicOff( SkeletalFinGSets( G_2 ) );
gap> DeactivateCachingOfCategory( SkeletalFinGSets( G_2 ) );
gap> DisableSanityChecks( SkeletalFinGSets( G_2 ) );
gap> DeactivateToDoList();
gap> ToM_1 := MatTom( TableOfMarks( G_1 ) );
[ [ 5, 0 ], [ 1, 1 ] ]
gap> k_1 := Size( ToM_1 );
2
gap> generating_set_1 := [ ];;
gap> for i in [ 1 .. k_1 ] do
>     M := ListWithIdenticalEntries( k_1, 0 )
>     ; M[i] := 1; Add( generating_set_1, FinGSet( G_1, M ) ); od;
gap> ToM_2 := MatTom( TableOfMarks( G_2 ) );
[ [ 20, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 10, 10, 0, 0, 0, 0, 0, 0, 0, 0 ],
  [ 10, 0, 10, 0, 0, 0, 0, 0, 0, 0 ],
  [ 10, 0, 0, 10, 0, 0, 0, 0, 0, 0 ],
  [ 5, 5, 5, 5, 5, 0, 0, 0, 0, 0 ],
  [ 4, 0, 0, 0, 0, 4, 0, 0, 0, 0 ],
  [ 2, 2, 0, 0, 0, 2, 2, 0, 0, 0 ],
  [ 2, 0, 2, 0, 0, 2, 0, 2, 0, 0 ],
  [ 2, 0, 0, 2, 0, 2, 0, 0, 2, 0 ],
  [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ] ]
gap> k_2 := Size( ToM_2 );
10
gap> generating_set_2 := [ ];;
gap> for i in [ 1 .. k_2 ] do
>     M := ListWithIdenticalEntries( k_2, 0 )
>     ; M[i] := 1; Add( generating_set_2, FinGSet( G_2, M ) ); od;
gap> SetInfoLevel( InfoWarning, 0 );
#@if IsPackageMarkedForLoading( "FinSetsForCAP", ">= 2018.09.17" )
gap> computed_group := ReconstructGroup(
>     SkeletalFinGSets( G_1 ),
>     HomSkeletalFinGSets,
>     ForgetfulFunctorSkeletalFinGSets( G_1 ),
>     generating_set_1,
>     EndAsEqualizer
> );;
```

```
gap> IsFinite( computed_group );;
gap> IsomorphismGroups( computed_group, G_1 ) <> fail;
true
gap> computed_group := ReconstructGroup(
>     SkeletalFinGSets( G_1 ),
>     HomSkeletalFinGSets,
>     ForgetfulFunctorSkeletalFinGSets( G_1 ),
>     generating_set_1,
>     EndByLifts
> );;
gap> IsFinite( computed_group );;
gap> IsomorphismGroups( computed_group, G_1 ) <> fail;
true
gap> computed_group := ReconstructGroup(
>     SkeletalFinGSets( G_2 ),
>     HomSkeletalFinGSets,
>     ForgetfulFunctorSkeletalFinGSets( G_2 ),
>     generating_set_2,
>     EndByLifts
> );;
gap> IsFinite( computed_group );;
gap> IsomorphismGroups( computed_group, G_2 ) <> fail;
true
#@fi
```

Bibliography

[BM18] Mohamed Barakat and Julia Mickisch, *FinSetsForCAP*, https://github.com/mohamed-barakat/FinSetsForCAP/ (Retrieved: 27 March 2018), 2017–2018. 39

[Bra17] Martin Brandenburg, *Einführung in die Kategorientheorie*, 2. Auflage, Springer Spektrum, Berlin, Heidelberg, 2017.

[end] *end in nLab*, https://ncatlab.org/nlab/show/end (Retrieved: 27 March 2018). 31

[GAP18] The GAP Group, *GAP – Groups, Algorithms, and Programming, Version 4.8.10*, 2018.

[GP] Sebastian Gutsche and Sebastian Posur, *Cap Manual*, https://github.com/homalg-project/CAP_project/raw/master/CAPManual.pdf (Retrieved: 27 March 2018).

[GSP18] Sebastian Gutsche, Øystein Skartsæterhagen, and Sebastian Posur, *The* CAP *project – Categories, Algorithms, Programming*, https://homalg-project.github.io/CAP_project, 2013–2018.

[Mic18] Julia Mickisch, *Die Realisierung elementarer Topoi auf dem Computer*, Universität Siegen, 2018. 3, 9, 11

[MZ18] Julia Mickisch and Fabian Zickgraf, *SkeletalGSetsForCAP*, https://github.com/zickgraf/SkeletalGSetsForCAP/ (Retrieved: 27 March 2018), 2017–2018.

[Tan] *Tannaka duality in nLab*, https://ncatlab.org/nlab/show/Tannaka+duality (Retrieved: 27 March 2018). 30, 31, 33

Index