Selecting Colimits for Parametrisation and Networks of Specifications

Till Mossakowski[1], Florian Rabe[2] and Mihai Codescu[3]

[1] Otto-von-Guericke-University of Magdeburg, Germany
[2] Jacobs University Bremen, Germany,
[3] Free University of Bozen-Bolzano, Italy

Abstract. Colimits are powerful tool for the combination of objects in a category. In the context of modeling and specification, they are used in the institution-independent semantics (1) of instantiations of parameterised specifications (e.g. in the specification language CASL), and (2) of combinations of networks of specifications (in the OMG standardised language DOL).

The problem of using colimits as the semantics of certain language constructs is that they are defined only up to isomorphism. However, the semantics of a complex specification in these languages is given by a signature and a class of models over that signature— not by an isomorphism class of signatures. This is particularly relevant when a specification with colimit semantics is further translated or refined. The user needs to know the symbols of a signature for writing a correct refinement.

Therefore, we study how to usefully choose one representative of the isomorphism class of all colimits of a given diagram. We develop criteria that colimit selections should meet. We work over arbitrary inclusive categories, but start the study how the criteria can be met with $\mathbb{S}et$-like categories, which are often used as signature categories for institutions.

1 Introduction

> "Given a species of structure, say widgets, then the result of interconnecting a system of widgets to form a super-widget corresponds to taking the *colimit* of the diagram of widgets in which the morphisms show how they are interconnected." [7]

Motivation The notion of colimit provides a natural way to abstract the idea that some objects of interest, which can be e.g. logical theories, software specifications or semiotic systems, are combined while taking into account the way they are related. Specification languages whose semantics involves colimits are CASL [16] (for instantiations of parameterised specifications) and its extension DOL (see [14,17] and `http://dol-omg.org`) (for combination of networks of specifications). Specware [24] provides a tool computing colimits of specifications that has been successfully used in industrial applications; [22] makes a strong case for the use of colimits in formal software development. The Heterogeneous Tool Set

(HETS, [13]) also supports computation of colimits, covering even the heterogeneous case [4]. Colimits have benn used for ontology alignment [25] and database integration [21]. Recently, colimits have provided the base mechanism for concept creation by blending existing concepts [10]. Moreover, colimits provide the basis for a good behaviour of parameterization in a specification language [6].[4]

The problem that arises naturally when using colimits is that they are not unique, but only unique up to isomorphism. By contrast, the semantics of a specification involves a specific signature, which must be selected from this isomorphism class. Also, any implementation of colimit computation in a tool must make an according choice of how the colimiting object actually looks, in particular when it comes to the names of its symbols. Otherwise, users have no control over the well-formedness of further specifications built from the colimit: Referring to symbols of the colimit is only possible with knowledge about the actual symbol names appearing in the colimit. To be useful in practice, it is desirable that such a choice appears natural to the user. For example, the names of original symbols should be preserved whenever possible.

Contribution Our contribution is two-fold. Firstly, we develop a suite of properties that can be used to evaluate and classify different colimit selections. All of these are motivated by the desire that parameterisation and combination of networks enjoy good properties. We show that these properties, although all desirable, cannot be realized at once. Secondly, we give solutions for systematically selecting colimits in various signature categories that provide good trade-offs between these conflicting properties.

Related work The semantics of CASL [1,12] provides some method for computation of specific pushouts. However, the chosen institutional framework (institutions with a lot of extra infrastructure) is rather complicated, while we use the much more natural framework of inclusion systems. Moreover, desired properties of pushouts are only discussed casually. Rabe [18] discusses three desirable properties of selected pushouts and conjectures that they are not reconcilable. We shed light on this conjecture and provide a total selection of pushouts, while [18] only provides a partial selection. The systematic investigation of selected colimits (i.e. beyond pushouts) is new to our knowledge. In the context of Specware, colimits are computed as equivalence classes [22]; however, this is quite cumbersome when dealing with real specifications.

Overview In Section 2, we recall some preliminaries as well as language constructs from CASL and DOL that involve colimits in their semantics. Then we develop criteria for elegant colimit selection in Section 3. In Section 4, we give colimit selections for various categories. We will also see that not every category admits a selection that satisfies all desirable properties. Therefore, we pursue a second goal in Section 5, namely to find useful categories for which we can give particularly elegant selections.

[4] When we use the term *specification*, our theory applies equally to *ontologies* and *models*, provided these have a formal semantics as theories of some institution.

2 Preliminaries

2.1 Categories with Symbols

The large variety of logical languages in use can be captured at an abstract level using the concept of *signature categories*. The objects of such a category are signatures which introduce syntax for the domain of interest, and the signature morphisms capture relations between signatures such as changes of notation, extensions, or translations. For example, signature categories feature heavily in the framework of *institutions* [8], where they are the starting point for abstractly capturing the semantics of logical systems and developing results independently of the specific features of a logical system.

In institutions and similar frameworks, the signature category is abstract, i.e., it is an arbitrary category. In practice, some properties of signature categories have emerged that are satisfied by the over-whelming majority of logical systems, and that are very helpful for establishing generic results.

For colimits, two properties are particularly important:

Definition 1 ([3]). *An **inclusive category** consists of a category* $\mathbf{C}$ *with a broad subcategory*[5] *that is a partially ordered class.*

*The morphisms of the broad subcategory are called **inclusions**, and we write* $A \hookrightarrow B$ *if there is an inclusion from* A *to* B*.*

In particular, $\mathbb{S}et$ is an inclusive category via the standard inclusions $A \hookrightarrow B$ iff $A \subseteq B$. Arbitrary categories can be recovered by using the identity relation as the partial order.

Definition 2. *An (inclusive)* category with symbols *consists of an (inclusive) category* $\mathbf{C}$ *and an (inclusion-preserving) functor* $|\_| : \mathbf{C} \to \mathbb{S}et$ *We call* $|A|$ *the set of symbols of* A*.*

In particular, $\mathbb{S}et$ is an inclusive category with symbols via $|A| = A$.

The intuition behind these definitions is that very often signatures can be seen as sets of named declarations. Then the subset relation defines the inclusion relation, and the names of the declarations define the set of symbols.

Signature categories are usually such that signatures that differ only in the choice of names are isomorphic. Then a key difficulty about colimits lies in selecting the set of names to be used in the colimit.

2.2 Specification Operators with Colimit Semantics

The power of the abstraction provided by institutions and related systems is best illustrated by the fact that languages like CASL and DOL provide syntax and semantics of specifications *in an arbitrary institution*. This is done by defining operators on specifications and morphisms.

[5] That is, with the same objects as $\mathbf{C}$.

A *basic specification* consists of a signature and a set of sentences[6]—called the *axioms*—over it. A kernel language of specification operators has been introduced in [19]. It includes union, renaming and hiding. CASL and DOL provide many further constructs.

The semantics of many of these operators can be defined as the colimit of a certain diagram. Therefore, such operators are often defined only up to isomorphism. In the sequel we recall important examples from CASL (parameterisation) and DOL (combination of networks).

Parametrisation Many specification languages, including CASL, allow specifications to be generic. A generic specification consists of a (formal) parameter specification P and a body specification B extending the formal parameter, i.e. $P \hookrightarrow B$. We make P explicit by writing $B[P]$.

A typical example is the specification $\mathit{List}[\mathit{Elem}]$ for lists parametrised by the specification Elem which declares a sort elem.

Given an actual parameter specification A and a specification morphism $\sigma : P \to A$, we write the instantiation of $B[P]$ with A via σ as $B[A \textbf{ fit } \sigma]$. Its semantics is given by the pushout on the left below:

$$\begin{tikzcd} P \arrow[r, hook] \arrow[d, "\sigma"] & B \arrow[d] \\ A \arrow[r, hook] & B[A \textbf{ fit } \sigma] \end{tikzcd} \quad \text{e.g.} \quad \begin{tikzcd} \mathit{Elem} \arrow[r, hook] \arrow[d, "\sigma"] & \mathit{List}[\mathit{Elem}] \arrow[d] \\ \mathit{Nat} \arrow[r, hook] & \mathit{List}[\mathit{Nat} \textbf{ fit } \mathit{elem} \mapsto \mathit{nat}] \end{tikzcd}$$

The right hand side above gives a typical example where the specification Nat declares a sort nat that is used to instantiate the sort elem.

A natural requirement is that the instantiated body $B[A \textbf{ fit } \sigma]$ extends the actual parameter A in much the same way as the body B extends the formal parameter P. For example, a sort *list* introduced in specification *List* should be kept (and not renamed) within the instantiation $\mathit{List}[\mathit{Nat} \textbf{ fit } \mathit{elem} \mapsto \mathit{nat}]$. Technically, this means that the semantics should not be an *arbitrary* colimit. Similarly, the user would expect that any symbols declared in the body should appear verbatim in the instantiated body, unless they have been renamed by σ.

Networks of Specifications In DOL, a network of specifications (called distributed specification in [15]) is a graph. Its nodes are labelled with pairs (O, SP) where SP is a specification and O its name. The edges are theory morphisms $(O_1, SP_1) \xrightarrow{\sigma} (O_2, SP_2)$, either induced by the import structure of the specifications, or by refinements.

A network is specified by giving a list of specifications O_i, morphisms M_i between themand sub-networks N_i, with the intuition that the graph of the network is the union of the graphs of all its elements.

`network N =`
$N_1, \ldots, N_m,$
$O_1, \ldots, O_n,$
$M_1, \ldots, M_p$

[6] It is straightforward but not essential here to make the notion of sentences precise.

Now the operator **combine** takes a network and produces the specification given by the colimit of the graph.

Example 1. In the example below, the network N3 consists of the nodes S, T2, and U2 and two automatically added edges, which are the inclusions from S to T2 and U2. Thus, N3 is a span, and **combine N3** yields its pushout. Indeed, both occurrences of sort s from S are identified in the pushout.

In the network N4, we exclude one of the automatically added inclusions. Thus, N4 is a graph with one isolated node for T2 and one inclusion edge from S to U2. **combine N4** yields the disjoint union of T2 and U2. That means that the two occurrences of sort s from S are kept seperate.

```
spec S = sort s end
spec T2 = S then sort t end
spec U2 = S then sort u end
network N3 = S, T2, U2 end
network N4 = N3 excluding S -> T2 end
```

3 Desirable Properties of Colimit Selections

The central definition regarding colimit selection is the following:

Definition 3. *Given a category* $\mathbf{C}$*, a* selection of colimits *is a partial function* $\boldsymbol{sel}$ *from* $\mathbf{C}$*-diagrams* D *to cocones on* D *such that* $\boldsymbol{sel}(D)$*, if defined, is a colimit for* D*. If* $\boldsymbol{sel}$ *is only defined for pushouts, we speak of a selection of pushouts, and so on.*

While it is trivial to give *some* selection of colimits (e.g., by using the axiom of choice or by randomly generating names), it turns out that selecting colimits *elegantly* is a non-trivial task. For example, selecting a colimit may require inventing new names, or there can be multiple conflicting strategies for selecting names. [18] conjectures that it is not possible to select pushouts in a way that the selected pushout are total, coherent, and enjoy natural names.

In this section, we introduce a suite of criteria for colimit selection.

3.1 Symbols of a Diagram

We work with an arbitrary inclusive category $\mathbf{C}$ with symbols. We are interested in selecting a colimit (C, μ_i) for a diagram $D : \mathbf{I} \to \mathbf{C}$. In most practically relevant signature categories, the construction of a colimit can be reduced to the construction of the colimit in $\mathbb{S}et$ of the corresponding sets of symbols. Because the colimit in $\mathbb{S}et$ amounts to taking a quotient of a disjoint union, we introduce the following auxiliary concept:

Definition 4 (Symbols of a Diagram). *Given a diagram* $D : \mathbf{I} \to \mathbf{C}$*, we define the set* $\mathbf{Sym}(D)$ *by*

$$\mathbf{Sym}(D) := \biguplus_{i \in |\mathbf{I}|} |D(i)| := \{(i, x) \mid i \in I, x \in |D(i)|\}.$$

Moreover, we define the preorder $\leq_D$ on $\mathbf{Sym}(D)$ *by*

$$(i,x) \ \leq_D \ (j, |D(m)|(x)) \text{ for any } m : i \to j \in \mathbf{I}.$$

and we define $\sim_D$ to be the least equivalence relation containing $\leq_D$.

Given any colimit (C, μ_i) of D, we embed $\mathbf{Sym}(D)$ *into $|C|$ by defining*

$$\mu_D(i,x) := |\mu_i|(x)$$

Intuitively, $\mathbf{Sym}(D)$ contains the symbols of all nodes of D. $\sim_D$ defines which symbols must definitely be identified in the colimit:

Proposition 1.

$$\sim_D \subseteq \ker(\mu_D)$$

Proof. This follows from μ being a cocone. ☐

In some categories such as $\mathbb{S}et$, we even have $\sim_D = \ker(\mu_D)$.

In principle, a natural property to desire of the selected colimit is that $|\mathbf{sel}(D)|$ is a quotient of $\mathbf{Sym}(D)$, in particular $|\mathbf{sel}(D)| = \mathbf{Sym}(D)/\sim_D$ if $\sim_D = \ker(\mu_D)$. However, that is often impractical, e.g., in the typical case where $|\Sigma|$ is intended to be a set of strings that serve as user-friendly names. In particular, we do not want to see the indices $i \in I$ creep into the symbol names in $|\mathbf{sel}(D)|$. Therefore, we define:

Definition 5 (Names and Name-Clashes). *For every equivalence class $X \in \mathbf{Sym}(D)/\sim_D$, let $\boldsymbol{Nam}(X) = \{x | (i,x) \in X\}$.*

We say that D is name-clash-free *if the sets $\boldsymbol{Nam}(X)$ are pairwise disjoint for all X. We say that D is* fully-sharing *if additionally all sets $\boldsymbol{Nam}(X)$ have size* 1*.*

Intuitively, name-clash-freeness means that whenever two nodes use the same symbol x, the diagram requires these two symbols to be shared in the colimit. A particularly common special case arises when both nodes import x from the same node. The following makes that precise:

Proposition 2. *Consider a diagram $D : I \to \mathbf{C}$. Assume that for all $(i,x), (j,x) \in \mathbf{Sym}(D)$ there are $(k,y) \in \mathbf{Sym}(D)$ and $m : k \to i$ and $n : k \to j$ in I such that $|m|(y) = |n|(y) = x$.*

Then D is name-clash-free. If additionally all edges in D are inclusions, D is fully-sharing.

The value of name-clash-freeness is the following: for the colimit, we can pick symbols that were already present in D. This allows selecting a colimit whose symbol names are inherited from the diagram (and thus already known to the user who requested te colimit). Moreover, if D is fully-sharing, these representatives are uniquely determined.[7]

[7] CASL has a mechanism of "compound identifiers" that ensures name-clash-freeness in multiple instantiations of parametrised specifications, such as $List[List[Elem]]$, see [16], p.47f. and p.224f.

3.2 Properties of Colimit Selections

Being thus prepared, we can now define a number of desirable properties that make a particular selection **sel** of colimits elegant.

The most obviously desirable property is that we select a colimit whenever we can:

Definition 6 (Completeness). ***sel*** *is complete if it is defined for every diagram that has a colimit.*

Choosing Symbols Typically, we cannot simply choose $|\mathbf{sel}(D)| = \mathbf{Sym}(D)/\sim_D$ because the choice of symbols is restricted:

Definition 7 (Name-Compliance). *Let* **Symbols** *be some subcategory of* $\mathbb{S}et$. *We call an object* Σ **Symbols**-*compliant if* $|\Sigma| \in$ **Symbols**. *A diagram is* **Symbols**-*compliant if all involved objects are.*

sel preserves **Symbols**-*compliance if* ***sel***(D) *is* **Symbols**-*compliant whenever* D *is.*

In practical systems, symbols must be chosen from a fixed set S, e.g., the set of alphanumeric strings. In that case, **Symbols** contains all sets that are subsets of S. If we want a compliance-preserving colimit selection, we have to pick names from S—that can be much more difficult to do canonically than to pick arbitrary symbols.

It is easy to select colimits by picking arbitrary symbols, e.g., by generating a fresh string as the name of any new declaration. But that is undesirable—it is preferable that the symbols of $\mathbf{sel}(D)$ are inherited from D in the following sense:

Definition 8 (Natural Names). ***sel*** *has* natural names *if for every name-clash-free diagram* D*, the selected colimit* ***sel***$(D) = (C, \mu_i)$ *is such that*

- $|C|$ *contains exactly one representative* $r \in$ ***Nam***(X) *for every equivalence class* X*,*
- $|\mu_i|$ *maps every* x *to the respective representative* r*.*

Note that if D is fully-sharing, natural names fully determine $|C|$. For the general case, we have to choose some r for each equivalence class. There are multiple options for making that choice canonical. For example:

Definition 9 (Origin-Based Names). *Let* ***sel*** *have natural names.*

sel *has* origin-*based symbol names if for every class* X *the chosen representative* r *is such that there is some* i *such that* (i, r) *is minimal in* X *with respect to* $\leq_D$.

Definition 10 (Majority). *Let* ***sel*** *have natural names.*

sel *has* majority-based *symbol names if for every class* X *the chosen representative* x *maximizes the cardinality of* $\{i \in I | (i, x) \in X\}$.

Accordingly, ***sel*** *has* majority-origin-based *symbol names if the above cardinality function is used to choose among multiple minimal elements.*

Example 2. Consider a span D

$$\begin{tikzcd} P \arrow[r, hook, "b"] \arrow[d, "a"] & B \\ A & \end{tikzcd}$$

We consider multiple situations given by the rows of the following table:

	$\lvert P\rvert$	$\lvert A\rvert$	$\lvert B\rvert$	$\lvert a\rvert$	$\lvert b\rvert$
1	$\{\}$	$\{x\}$	$\{x\}$		
2	$\{p\}$	$\{a, a'\}$	$\{b, b'\}$	$p \mapsto a$	$p \mapsto b$
3	$\{p, p'\}$	$\{a\}$	$\{p, p'\}$	$p \mapsto a, p' \mapsto a$	$p \mapsto p, p' \mapsto p'$
4	$\{elem\}$	$\{elem, list\}$	$\{nat, +\}$	$elem \mapsto elem$	$elem \mapsto nat$

Depending on the situation, different colimit selections are possible:

1. The diagram is not name-clash-free, and we cannot inherit names.
2. The diagram is name-clash-free but not fully sharing. The sets $\mathbf{Nam}(\_)$ are $\{p, a, b\}$, $\{a'\}$, and $\{b'\}$. Thus, there are three possible colimits that have natural names. All three satisfy the majority condition. The origin condition allows uniquely selecting $|\mathbf{sel}(D)| = \{p, a', b'\}$.
3. The only set $\mathbf{Nam}(\_)$ is $\{p, p', a, p, p'\}$ (where we repeat elements to indicate how often they occur in the corresponding equivalence class). We can have natural names, but neither majority nor origin yield a unique choice.
4. This is a typical case of instantiating a parametric specification (here: lists with a parameter for the type of elements) with an actual parameter (here: the set of natural numbers). The sets $\mathbf{Nam}(-)$ are $\{elem, elem, nat\}$, $\{list\}$, and $\{+\}$. We can have natural names, and both origin and majority uniquely yield $|\mathbf{sel}(D)| = \{elem, +, list\}$. However, neither is elegant: The desired choice would be $\{nat, +, list\}$.

Pushouts along Inclusions Pushouts along inclusions are of particular importance because they provide the semantics of parametrization. As in Section 2.2, D is a diagram as given on the right.

$$\begin{tikzcd} P \arrow[r, hook] \arrow[d, "\sigma"] & B \\ A & \end{tikzcd}$$

The following property is motivated by the desire that instantiating parameterised specifications should always be defined:

Definition 11 (Total pushouts). ***sel*** *has* total pushouts *if it is defined for all spans where one arrow is an inclusion.*

Moreover, it is desirable that the instantiation extends A in the same way in which P extends A. The following definitions make this precise:

Definition 12 (Pushout-stable Inclusions). *Let* ***sel*** *have total pushouts.*

sel *has* pushout-stable *inclusions if the pushout selection preserves the inclusion, i.e.,* $\boldsymbol{sel}(D)$ *is of the form*

$$\begin{tikzcd} P \arrow[r, hook] \arrow[d, "\sigma"] & B \arrow[d, "\sigma^B"] \\ A \arrow[r, hook] & \sigma(B) \end{tikzcd}$$

Definition 13 (Pushout-Stable Names). *Assume* **C** *has symbols, and let* ***sel*** *have pushout-stable inclusions.*

sel *has* pushout-stable names *if for every selected pushout*

$$\begin{tikzcd} P \arrow[r, hook] \arrow[d, "\sigma"] & B \arrow[d, "\sigma^B"] \\ A \arrow[r, hook] & \sigma(B) \end{tikzcd} \qquad \begin{tikzcd} |P| \arrow[r, hook] \arrow[d, "|\sigma|"] & |B| \arrow[d, "|\sigma^B|"] \\ |A| \arrow[r, hook] & |\sigma(B)| \end{tikzcd}$$

we have $|\sigma(B)| \setminus |A| = |B| \setminus |P|$ *and* $|\sigma^B|$ *is the identity on that set.*

The aim of pushout-stable inclusions is that we can have
- (vertically) $\_^B$ as a functor $(P \downarrow \mathbf{C}) \to (B \downarrow \mathbf{C})$,
- (horizontally) $\sigma(\_)$ as functor $(P \downarrow \mathbf{C}) \to (A \downarrow \mathbf{C})$ mapping extensions of P to extensions of A.

However, in general, the functoriality laws only hold up to isomorphism. Therefore, we want to impose an additional condition, which is adapted from [18]:

Definition 14 (Coherent Pushouts). *Let* ***sel*** *have pushout-stable inclusions. Then* ***sel*** *has* coherent pushouts *if the following coherence conditions hold:*

1. $id_P(B) = B$ *and* $id_P^B = id_B$,
2. $\sigma(P) = A$ *and* $\sigma^P = \sigma$,
3. $(\sigma_1;\sigma_2)(B) = \sigma_2(\sigma_1(B))$ *and* $(\sigma_1;\sigma_2)^B = \sigma_1^B;\sigma_2^{\sigma_1(B)}$ *and finally*
4. *for* $P \hookrightarrow B_1 \hookrightarrow B_2$, $\sigma(B_2) = \sigma^{B_1}(B_2)$ *and* $\sigma^{B_2} = (\sigma^{B_1})^{B_2}$

where two conditions refer to the following diagrams

$$\begin{tikzcd} P \arrow[r, hook] \arrow[d, "\sigma_1"] & B \arrow[d, "\sigma_1^B"] \arrow[ddr, "(\sigma_1;\sigma_2)^B"] & \\ A \arrow[r, hook] \arrow[d, "\sigma_2"] & \sigma_1(B) \arrow[d, "\sigma_2^{\sigma_1(B)}"] & \\ A' \arrow[r, hook] & \sigma_2(\sigma_1(B)) \arrow[r, equal] & (\sigma_1;\sigma_2)(B) \end{tikzcd} \qquad \begin{tikzcd} P \arrow[r, hook] \arrow[d, "\sigma"] & B_1 \arrow[r, hook] \arrow[d, "\sigma^{B_1}"] & B_2 \arrow[d, "\sigma^{B_2}"] \arrow[dr, "(\sigma^{B_1})^{B_2}"] & \\ A \arrow[r, hook] & \sigma(B_1) \arrow[r, hook] & \sigma(B_2) \arrow[r, equal] & \sigma^{B_1}(B_2) \end{tikzcd}$$

and ensure that pushouts compose vertically and horizontally.

Coherence The coherence conditions for pushouts can be generalized to arbitrary diagrams. The general idea is that if there are multiple ways to construct a colimit step-by-step, then it should not matter in which order the construction proceeds. Here step-by-step means that we first construct a colimit of a subdiagram of D and then add that colimit to D and construct a colimit of the resulting bigger diagram, and so on.

A formal definition for the general case is rather difficult. The following special case is adapted from [2]:

Definition 15 (Interchange). ***sel*** *has* interchange *if given a name-clash-free diagram $D : \mathbf{I} \times \mathbf{J} \to \mathbf{C}$ (seen as a bifunctor) involving inclusions only*

$$\boldsymbol{sel}_{i\in\mathbf{I}}(\boldsymbol{sel}_{j\in\mathbf{J}}D(i,j)) = \boldsymbol{sel}_{j\in\mathbf{J}}(\boldsymbol{sel}_{i\in\mathbf{I}}D(i,j))$$

With an isomorphism instead of equality, this condition always holds.

To state the coherence condition in full generality, we need a few auxiliary definitions:

Definition 16. *Consider a category I with an object i such that every I-object has at most one arrow into i.*

We write $I \setminus i$ for the subcategory of I formed by removing i. We write $I^{\to i}$ for the subcategory of I formed by removing i and all nodes that have no arrow into i. For a diagram $D : I \to \mathbf{C}$, we write $D \setminus i$ and $D^{\to i}$ for the corresponding restrictions of D.

We say that i is a colimit node *of D if $D(i)$ and the set of all morphisms $D(m)$ for I-arrows m into i are a colimit of $D^{\to i}$. If additionally that colimit is equal to* ***sel***$(D^{\to i})$*, we call i a* ***sel****-colimit node.*

The intuition behind colimit nodes is that they arise by taking a colimit of a subdiagram and can be ignored when forming a colimit of the entire diagram. For example, in the two commuting diagrams of Def. 14, the nodes $\sigma_1(B)$ and $\sigma(B_1)$ are colimit nodes. They arise as the intermediate results of constructing the pushout in two steps. In general, they arise when constructing a colimit step-by-step:

Proposition 3. *Consider a diagram $D : I \to \mathbf{C}$ with a colimit node i. Then D and $D \setminus i$ have the same colimits.*

Proof. For every D-colimit we obtain a $D \setminus i$-cone by removing the injection from i. Vice versa, every $D \setminus i$-colimit (C, μ) can be uniquely extended to a D-cone with the unique factorization $\mu_i : D(i) \to C$ for the colimit $D(i)$.

In both cases, the colimit properties are shown by diagram chase. □

Now we can define that coherence means that we can indeed ignore colimit nodes when selecting a colimit:

Definition 17. ***sel*** *is* coherent *for the diagram D if for every* ***sel****-colimit node i we have that* ***sel***(D) *and* ***sel***$(D \setminus i)$ *are equal (apart from the former additionally containing the uniquely determined injection μ_i).*

By iterating the coherence property, we can remove or add **sel**-colimit nodes from/to a diagram without affecting the selected colimit.

4 Colimit Selections for Typical Signature Categories

4.1 Sets

As the simplest possible signature category, we consider the category $\mathbb{S}et$ (with standard inclusions and the identity symbol functor).

We first provide a positive result: We can realise several desirable properties at once:

Theorem 1. *$\mathbb{S}et$ has a selection of colimits that has completeness, pushout-stable inclusions, total pushouts and interchange.*

Moreover, for name-clash-free diagrams, this selection has natural names, pushout-stable names, coherent pushouts.

Second, we give a negative result: There is a small set of desirable properties that cannot be realised at once:

Theorem 2. *$\mathbb{S}et$ does not have a selection that has total pushouts, pushout-stable inclusions and names, and coherent pushouts.*

Thm. 1 shows that in $\mathbb{S}et$, we can realise several criteria for colimit selection we have defined so far. However, in the construction in the proof of Thm. 1, all colimits not being pushouts along inclusions nor colimits of diagrams using inclusions only are selected randomly. This is unsatisfactory, because for these colimits, our goal that the user has control over names has not been reached.

Indeed, origin and majority can contradict the principles that we have introduced so far:

Proposition 4. *The selection constructed in Thm. 1 does not satisfy the origin and majority principles.*

Proof. Consider a span $B \stackrel{\iota}{\hookleftarrow} P \stackrel{\sigma}{\to} A$ with σ not an inclusion. Then in Thm. 1, $\sigma(B) := A \cup (B \setminus A) \cup B'$. This means any symbol from P that is renamed by σ will not appear in the pushout object $\sigma(B)$, contradicting the origin principle. Moreover, because $P \subseteq B$, such a symbol will occur twice (with different objects) in its equivalence class, but the equivalent symbol from A (occurring only once) is selected in the pushout. $\square$

A closer inspection shows that that pushout-stable inclusions and names contradict the origin and majority property. Moreover, it is evident that origin and majority can contradict each other. Consider e.g.

$$\begin{tikzcd}
\{a\} \arrow[r] \arrow[d] & \{b\} \arrow[d] \\
\{b\} \arrow[r] & \{x\}
\end{tikzcd}$$

Origin would lead to $x = a$, while majority would lead to $x = b$.

Nevertheless, the origin and majority are useful principles that can guide pushout selection in cases where the other principles are not be able to do this.

Proposition 5. *The selection constructed in Thm. 1 can be modified to have majority-origin natural names.*

4.2 Product Categories

Signatures of many logical systems of practical interest are often tuples of sets of symbols of different kind. For example, OWL signatures consist of sets of atomic classes, individuals, object and data properties. To be able to transfer the selection of colimits and its properties defined for $\mathbb{S}et$ to categories of tuples of sets, we make use of a more general result that ensures that selection of colimits and its properties are stable under products.

Theorem 3. *Let $(\mathbf{C}_j)_{j \in J}$ be a family of inclusive categories with symbols and assume selections of colimits sel_j that have the properties in Thm. 1 or Prop. 5. Then the product $\Pi_{j \in J} C_j$ can be canonically turned into a inclusive category with symbols that also has a selection of colimits **sel** with the same properties.*

Example 3. In the case of multi-sorted logics with function or predicate symbols, we can define a selection function for colimits in a step-wise manner. Let us consider the case of multi-sorted equational logic, that we denote EQL. If we fix a set of sorts S, let $B(S) = \mathbf{Sign}_S^{EQL}$ be the category of multi-sorted algebraic signatures with sort set S. We then can express this category as

$$\mathbf{Sign}_S^{EQL} = \Pi_{w \in S^*, s \in S}\ \mathbb{S}et.$$

Objects of this category provide a set of operation symbols $F_{w,s}$ for each string of argument sorts w and result sort s. With the canonical lifting of the symbol functors of the factors (all of which are the identity on $\mathbb{S}et$) to this product, we obtain the symbol functor on $\mathbf{Sign}_S^{EQL}$ given by $|\_| = \biguplus_{i \in J} |\pi_j(\_)|$, which decorates each operation symbol with argument and result sorts. We write $f : w \to s \in |F|$ instead of $((w, s), f) \in |F|$.

4.3 Split Fibrations

Thm. 3 gives us a selection of colimits for $\mathbf{Sign}_S^{EQL}$. However, our overall goal is to provide such a selection for $\mathbf{Sign}^{EQL}$. Now $\mathbf{Sign}^{EQL}$ is a split fibration $\mathbf{Sign}^{EQL} \to \mathbb{S}et$, with fibres $\mathbf{Sign}_S^{EQL}$. It is well-known that a split fibration can be obtained as Grothendieck construction (flattening) of an indexed category indexing the fibres. Hence, we will construct such an indexed category for EQL. This is achieved by observing that each function $u : S \to S'$ leads to a functor $B_u : \mathbf{Sign}_{S'}^{EQL} \to \mathbf{Sign}_S^{EQL}$ defined as $B_u(F') = F$, where $F_{w,s} = F'_{u(w),u(s)}$. This functor has a left adjoint denoted $L_u : \mathbf{Sign}_S^{EQL} \to \mathbf{Sign}_{S'}^{EQL}$ defined as $L_u(F) = F'$, where $F'_{w',s'} = \cup_{w \in S^*, s \in S, u(w) = w', u(s) = s'} F_{w,s}$.

We thus obtain an indexed inclusive category $B : \mathbb{S}et^{op} \to ICat$, and it suffices to show that the selection of colimits and its properties are stable under the Grothendieck construction (flattening, see [23]).

Theorem 4. *Let $B : \mathbf{Ind}^{op} \to \mathbb{IC}at$ be an indexed inclusive category (where* **Ind** *is inclusive itself) such that*

- *B is* locally reversible*, i.e. for each $u : i \to j$ in* **Ind***, $B_u : B_j \to B_i$ has a selected left adjoint $F_u : B_i \to B_j$ (note that we do not require coherence of the F_u),*
- **Ind** *has a selection of colimits* $\boldsymbol{sel}_{\mathbf{Ind}}$*,*
- *each category B_i has a selection of colimits $\boldsymbol{sel}^i$, for $i \in |\mathbf{Ind}|$.*

Then the Grothendieck category $B^{\#}$ is itself an inclusive category.[8]

Theorem 5. *Under the assumptions of Thm. 4, let $(|\_|\theta) : B \to \mathbb{I}nd\mathbb{S}et$ be a (faithful inclusive) oplax indexed functor (where $\mathbb{I}nd\mathbb{S}et : \mathbf{Ind}^{op} \to \mathbb{C}at$ is the constant functor delivering $\mathbb{S}et$).*

This amounts to, for each B_i, a (faithful inclusive) symbol functor $|\_|_i : B_i \to \mathbb{S}et$, and for each $u : i \to j$, $\theta_u : B_u; |\_|_i \to |\_|_j$ a natural transformation, such that the θ_u are coherent.

Then $B^{\#}$ can be equipped with a symbol functor as well.

Proof. Define $|(i, A_i)| = |i| \uplus |A_i|_i$, and $|(u : i \to j, \sigma)| = |u| \uplus (|\sigma|_i; (\theta_u)_{A_j})$. $\square$

Theorem 6. *Under the assumptions of Thm. 4 and Thm. 5, extended by:*

- *F_u preserves inclusions, and moreover,*
- *the unit and counit of the adjunction are inclusions.*

If **Ind** *and each B_i have colimit selections enjoying the properties of Thm. 1, then so does $B^{\#}$.*

We can apply Thm. 6 to $B : \mathbb{S}et^{op} \to \mathbb{I}\mathbb{C}at$ as defined above to obtain a selection of colimits $\mathbf{sel}^{EQL}$ for EQL signatures. By the theorem, $\mathbf{sel}^{EQL}$ has the properties in Thm. 1.

Example 4. We apply these result to EQL, where $B_s = \mathbf{Sign}_S^{EQL}$, using the symbol functors $|\_|_S : \mathbf{Sign}_S^{EQL} \to \mathbb{S}et$ ($S \in |\mathbb{S}et|$) defined above. Given $u : S \to S'$, $\theta_u : B_u; |\_|_S \to |\_|_{S'}$ is defined as $(\theta_u)_{F'} : |B_u(F')| \to |F'|$, acting as $(\theta_u)_{F'}(f : u(w) \to u(s)) = f : w \to s$. Using Thm. 5, we obtain the usual symbol functor for many-sorted signatures, which for any signature delivers the set of sorts plus the set of typed function symbols of form $f : w \to s$.

Again, the symbol selection principles of Prop. 5 carry over.

5 Categories for Improved Colimit Selection

5.1 Named Specifications

An important technique for avoiding name clashes is to use two-partite IRIs as symbols. These symbols consists of

[8] Note that this construction extends to institutions, yielding Grothendieck institutions, see [5].

– **namespace**: an IRI that identifies the containing specification, usually ending with `/`[9]
– **local name**: a name (not containing `/`) that identifies a non-logical symbol within a specification.

Here IRIs are Internationalized Resource Identifiers for identification per IETF/RFC 3987:2005. Let **IRI** be the subcategory of $\mathbb{S}et$ containing only the sets of bipartite IRIs.

For most practical purposes, it is acceptable to restrict attention to **IRI**-compliant signatures. For example, DOL (in accordance with many other languages) strongly recommends using bipartite IRIs.

Note that in an **IRI**-compliant signature Σ, the symbols in $|\Sigma|$ may have different namespaces. For example, in DOL, namespaces M serve as the identifiers of basic specification Σ, and then symbols in $|M|$ are of the form M/sym. But when a specification N imports M, (see Sect. 2.2), the namespace M of the imported symbols is retained and only new symbols declared in N use the namespace N.

The main advantage of using IRIs is that specifications (and thus the symbols in them) have globally unique names [9]. That makes name clashes much less common:

Proposition 6. *Consider a set of basic signatures with pairwise different namespaces. Then diagrams generated by networks consisting only of* **IRI**-*compliant basic specifications and imports are fully-sharing.*

Proof. Because basic specifications have unique identifiers, the result follows immediately from Prop. 2. □

In practice, the assumptions of Prop. 6 quite often hold, because networks to be combined often consist of import links only.

Proposition 7. *Consider* $\mathbb{S}et$ *with standard inclusions and the identity symbol functor. The selection constructed in Prop. 5 can be modified to a selection that additionally preserves* **IRI**-*compliance.*

Proof. We just need to ensure that new symbols in the colimit are of the form N/sym for some fresh namespace N. □

However, generating fresh namespaces interacts poorly with coherence.

5.2 Structured Symbol Names

There are essentially two problems when trying to select colimits canonically: name clashes and ambiguous names. Intuitively, name clashes arise if we have one name for multiple symbols. And ambiguity arises if we have multiple names

[9] In some languages, `#` is used instead of /. But this has the disadvantage that, when used as an IRL, the fragment following the `#` is not transmitted to servers.

for one symbol. If neither is the case, named specifications are usually sufficient to obtain canonical colimits.

Our goal now is to handle on name clashes and ambiguity. We introduce a subcategory **Vocab** of $\mathbb{S}et$ and focus on **Vocab**-compliance-preserving colimits. We want to pick **Vocab** in such a way that we can select canonical colimits elegantly.

To motivate the following definition of **Vocab**, let us look again at the causes behind name clashes and ambiguity. Name clashes arise if the same node occurs multiple times in a diagram. For example, consider two nodes i and j (without any arrows) and $|D(i)| = \{a\}$ and $|D(j)| = \{a\}$. (This occurs, for example, when taking the disjoint union of the set $\{a\}$ with itself.) Because this diagram is not name-clash-free, we cannot have natural names in the colimit. Our solution below introduces qualifiers that create two copies p/a and q/a of the clashing name a.

Ambiguity arises if a diagram contains a non-inclusion arrow. For example, consider $m : i \to j$, and $|D(i)| = \{a\}$ and $|D(j)| = \{b\}$ and $|D(m)|(a) = b$. $\sim_D$ has one equivalence class, which contains (i, a) and (j, b). In Section 3.2, we focused on choosing either a or b as a natural name in the colimit. Our solution here retains both names and chooses the set $\{a, b\}$ as a symbol in the colimit.

Because colimits can be iterated, **Vocab** must allow for any combination of those two constructions. That yields the following definition:

Definition 18 (Structured Symbols). *We assume a fixed set* **Name** *of strings (which we call* ***names****).*

We write **QualName** *for the set of lists of names (which we call* ***qualified names****). We assume $Name \subset QualName$, and we write* **nil** *for the empty list and p/q for the concatenation of lists.*

A ***structured symbol*** *is a set of qualified names.*

A ***vocabulary*** *V is a set of pairwise disjoint structured symbols. We write $\overline{V}$ for $\bigcup_{S \in V} S$, and for every $s \in \overline{V}$ we write $[s]$ for the unique $S \in V$ such that $s \in S$.*

We write **Vocab** *for the full subcategory of $\mathbb{S}et$ containing only the vocabularies.*

The operation $[s]$ is crucial: It allows us to use any $s \in S \in V$ as a representative for S. Thus, in order to use structured symbols, we do not have to change our external (human-facing) syntax: Users can still write and read s. We only have to change our internal (machine-facing) syntax by maintaining the set S.

Above we left open the question where the qualifiers come from that we use to disambiguate name clashes. We will assume that these are given by the user by assigning labels to some nodes in the diagram:

Definition 19 (Labeled Diagram). *A labeled* $\mathbf{C}$*-diagram (D, L) consists of a diagram $D : I \to \mathbf{C}$ and a function L from I-objects to* $\mathbf{Name} \cup \{\mathbf{nil}\}$*.*

L can be a partial function because we only need to label those nodes that are involved in name clashes. However, it is more convenient to make L a total function by assuming that all unlabeled nodes are labeled with the empty list **nil**.

Similar to Def. 4, we define the symbols of a labeled diagram:

Definition 20 (Symbols of a Labeled Diagram). *Let* $(D : I \to \mathbf{Vocab}, L)$ *be a labeled diagram over* **Vocab***. We define:*

$$\mathbf{Sym}(D, L) = \{(i, L(i)/x) \mid i \in I, x \in \overline{D(i)}\}$$

$(i, L(i)/x) \leq_{DL} (j, L(j)/y)$ *if for some* $m : i \to j \in \mathbf{I}, D(m)(S) = T, x \in S, y \in T$

$\sim_{DL}$ *is the equivalence relation on* $\mathbf{Sym}(D, L)$ *generated by* $\leq_{DL}$*.*

For every $X \in \mathbf{Sym}(D, L)/\sim_{DL}$*, let* $\boldsymbol{Nam}(X) = \{q : (i, q) \in X\}$*. We say that* (D, L) *is* name-clash-free *if the sets* $\boldsymbol{Nam}(X)$ *are pairwise disjoint.*

Every plain diagram can be seen as a labeled diagram by using $L(i) = \mathbf{nil}$ for all i. In that case, the definition of name-clash-free of Def. 20 coincides with the one from Def. 5.

We can now see the power of structured symbols by giving a selection of colimits in **Vocab**:

Theorem 7 (Colimits of Vocabularies). *Let* (D, L) *be a name-clash-free labeled diagram. Then*

- *the set* $\boldsymbol{sel}(D, L)$ *defined by* $\{\boldsymbol{Nam}(X) : X \in \mathbf{Sym}(D, L)/\sim\}$ *is a vocabulary,*
- *the maps* $\mu_i : D(i) \to \boldsymbol{sel}(D, L)$ *defined by* $\mu_i([x]) = [L(i)/x]$ *are well-defined.*

Then $\boldsymbol{sel}(D, L)$ *and the* μ_i *form a colimit of* D*.*

sel does not exactly have the desirable properties described in Section 3.2. But it has variants of them that are good trade-offs:

- **sel** is complete in the sense that labels can be added to any diagram to obtain name-clash-freeness.
- **sel** reduces to union for name-clash-free unlabeled diagrams of inclusions (and therefore satisfies interchange).
- **sel** has pushout-stable inclusions for name-clash-free unlabeled diagrams.
- **sel** has natural names in the sense that $L(i)/x$ can be used to identify the corresponding symbol in the colimit, and every symbol in the colimit is of that form.
- **sel** is coherent for all labeled diagrams in which all **sel**-colimit nodes are unlabeled.

6 Conclusion

We have provided some useful principles for colimit selection, and studied how far these principles can be actually realised. Some principles contradict each other, so they need to be prioritised. The overall goal is to give the user as much control and predictability over names as possible. This is particularly important for languages such as CASL and DOL, providing powerful constructs for both parameterisation and combination of networks, realised through colimits. We have shown that our results are stable under products and Grothendieck constructions; hence they carry over to more complex signature categories like many-sorted logics, HasCASL [20] (without subsorts) or even categories of heterogeneous specification (which usually are also obtained via a Grothendieck construction).

While we have worked with $\mathbb{S}et$ and $\mathbb{S}et$-like categories, future work should extend the results to more complex categories. E.g. the signature category of the subsorted CASL logic cannot be obtained from $\mathbb{S}et$ by products and indexing; instead some quotient construction is needed [11]. Another open question is whether coherence for pushouts can usefully be generalised to other types of colimit.

One important motivation for this work has been the need to obtain a better theory for the implementation of colimits in Hets. Currently, the implementation follows the majority principle only, which led to complaints from the user community, especially from the Coinvent project using colimits for conceptual blending. In the future, this will be revised according to the results of this paper.

References

1. H. Baumeister, M. Cerioli, A. Haxthausen, T. Mossakowski, P. D. Mosses, D. Sannella, and A. Tarlecki. CASL Semantics. In Peter D. Mosses, editor, *CASL Reference Manual*, volume 2960 of *Lecture Notes in Computer Science*, part ÏII. Springer Verlag, London, 2004. Edited by D. Sannella and A. Tarlecki.
2. F. Borceux. *Handbook of Categorical Algebra I – III.* Cambridge University Press, 1994.
3. Virgil Emil Căzănescu and Grigore Roşu. Weak inclusion systems. *Mathematical Structures in Computer Science*, 7(2):195–206, April 1997.
4. M. Codescu and T. Mossakowski. Heterogeneous colimits. In F. Boulanger, C. Gaston, and P.-Y. Schobbens, editors, *MoVaH'08 Workshop*, 2008.
5. R. Diaconescu. *Institution-independent Model Theory.* Birkhäuser Basel, 2008.
6. R. Diaconescu, J.A. Goguen, and P. Stefaneas. Logical Support for Modularisation. In *2nd Workshop on Logical Environments*, pages 83–130. CUP, New York, 1993.
7. J. A. Goguen. A categorical manifesto. *Mathematical Structures in Computer Science*, 1:49–67, 1991.
8. J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39:95–146.
9. W3C SEMANTIC WEB DEPLOYMENT WORKING GROUP. Best practice recipes for publishing rdf vocabularies. *W3C Working Group Note, 28 August 2008.* *http://www.w3.org/TR/2008/NOTE-swbp-vocab-pub-20080828/*, 2008.
10. O. Kutz, J. Bateman, T. Mossakowski, F. Neuhaus, and M. Bhatt. E pluribus unum - formalisation, use-cases, and computational support for conceptual blending. In T. R. Besold, M. Schorlemmer, and A. Smaill, editors, *Computational Creativity Research: Towards Creative Machines*, volume 7 of *Atlantis Thinking Machines*, pages 167–196. Atlantis Press, 2015.
11. T. Mossakowski. Colimits of order-sorted specifications. In F. Parisi Presicce, editor, *Proc. 12th WADT*, volume 1376 of *LNCS*, pages 316–332. Springer, 1998.

12. T. Mossakowski. Specification in an arbitrary institution with symbols. In C. Choppy, D. Bert, and P. Mosses, editors, *WADT 1999*, volume 1827 of *LNCS*, pages 252–270. Springer Verlag, London, 2000.
13. T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editors, *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522. Springer, Heidelberg, 2007.
14. Till Mossakowski, Oliver Kutz, Mihai Codescu, and Christoph Lange. The distributed ontology, modeling and specification language. In Chiara Del Vescovo, Torsten Hahmann, David Pearce, and Dirk Walther, editors, *WoMo 2013*, volume 1081 of *CEUR-WS online proceedings*, 2013.
15. Till Mossakowski and Andrzej Tarlecki. Heterogeneous logical environments for distributed specifications. In Andrea Corradini and Ugo Montanari, editors, *WADT 2008*, number 5486 in Lecture Notes in Computer Science, pages 266–289. Springer, 2009.
16. Peter D. Mosses, editor. *CASL Reference Manual*. Number 2960 in LNCS. Springer Verlag, 2004.
17. Object Management Group. The distributed ontology, modeling, and specification language (DOL), 2015. OMG draft standard available at http://www.omg.org/spec/DOL/.
18. F. Rabe. How to Identify, Translate, and Combine Logics? *Journal of Logic and Computation*, 2014. doi:10.1093/logcom/exu079.
19. D. Sannella and A. Tarlecki. Specifications in an arbitrary institution. *Information and Computation*, 76:165–210, 1988.
20. Lutz Schröder and Till Mossakowski. Hascasl: Integrated higher-order specification and program development. *Theoret. Comp. Sci.*, 410(12-13):1217–1260, 2009.
21. Patrick Schultz, David I. Spivak, Christina Vasilakopoulou, and Ryan Wisnesky. Algebraic databases. *CoRR*, abs/1602.03501, 2016.
22. Douglas R. Smith. Composition by colimit and formal software development. In Kokichi Futatsugi, Jean-Pierre Jouannaud, and José Meseguer, editors, *Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*, volume 4060 of *Lecture Notes in Computer Science*, pages 317–332. Springer, 2006.
23. Andrzej Tarlecki, Rod M. Burstall, and Joseph A. Goguen. Some fundamental algebraic tools for the semantics of computation: Part 3: Indexed categories. *Theor. Comput. Sci.*, 91(2):239–264, 1991.
24. K. E. Williamson, M. Healy, and R. A. Barker. Industrial applications of software synthesis via category theory-case studies using Specware. *Autom. Softw. Eng*, 8(1):7–30, 2001.
25. A. Zimmermann, M. Krötzsch, J. Euzenat, and P. Hitzler. Formalizing Ontology Alignment and its Operations with Category Theory. In *Proc. of FOIS-06*, pages 277–288, 2006.

A Omitted Proofs

Theorem 1. $\mathbb{S}et$ (with standard inclusions and the identity symbol functor) has a selection of colimits that has completeness, pushout-stable inclusions, total pushouts and interchange. Moreover, for name-clash-free diagrams, this selection has natural names, coherent pushouts and pushout-stable names.

Proof. If name-clash-freeness is satisfied and the diagram consists of inclusions only, just take the union as colimit, i.e. $C = \bigcup_{i \in |I|} D(i)$. This shows that interchange holds.

Given a span $B \overset{\iota}{\hookleftarrow} P \overset{\sigma}{\to} A$ with σ not an inclusion, let $\sigma(B) := A \cup (B \setminus A) \cup B'$, where $\kappa : B' \cong (B \cap A) \setminus P$ such that $B' \cap (A \cup (B \setminus A)) = \emptyset$. Define

$$\begin{tikzcd} P \arrow[r, hook, "\iota"] \arrow[d, "\sigma"] & B = P \cup (B \setminus P) \arrow[d, "\sigma^B = \sigma \cup \theta"] \\ A \arrow[r, hook] & \sigma(B) = A \cup ((B \setminus (P \cup A)) \cup B') \end{tikzcd}$$

where $\theta : B \setminus P \to (B \setminus (P \cup A)) \cup B'$ is given by

$$\theta(x) = \begin{cases} \kappa^{-1}(x), & \text{if } x \in (B \cap A) \setminus P \\ x, & \text{if } x \in B \setminus (P \cup A) \end{cases}.$$

Suppose there is any cocone $(C, \nu_A : A \to C, \nu_B : B \to C)$. The mediating morphism $c : \sigma(B) \to C$ is defined as

$$\begin{cases} \nu_A(x), & \text{if } x \in A \\ \nu_B(x), & \text{if } x \in B \setminus A \\ \nu_B(\kappa^{-1}(x)), & \text{if } x \in B' \end{cases}$$

This shows that inclusions are pushout-stable, and total pushouts exist. Moreover, if name-clash-freeness holds for the span $B \overset{\iota}{\hookleftarrow} P \overset{\sigma}{\to} A$, $B' = \emptyset$, hence θ is the identity and we have natural names for pushouts. Using the notation of the coherence diagrams, vertical coherence of pushouts can be shown as follows: by name-clash-freeness, $(B \cap A) \setminus P = \emptyset$, hence $B' = \emptyset$ and $\sigma_1(B) = A \cup (B \setminus (P \cup A))$. Similarly, $\sigma_2(A \cup (B \setminus (P \cup A))) = A' \cup ((A \cup (B \setminus (P \cup A))) \setminus (A \cup A')) = A' \cup (B \setminus (P \cup A \cup A'))$. On the other hand, $(\sigma_1; \sigma_2)(B) = A' \cup (B \setminus (P \cup A')$. But since $(B \cap A) \setminus P = \emptyset$, $B \setminus (P \cup A \cup A') = B \setminus (P \cup A')$, hence $\sigma_2(\sigma_1(B)) = (\sigma_1; \sigma_2)(B)$. Concerning horizontal coherence, $\sigma^{B_1}(B_2) = \sigma(B_1) \cup (B_2 \setminus (B_1 \cup \sigma_1(B))) = A \cup (B_1 \setminus (P \cup A)) \cup (B_2 \setminus (B_1 \cup A \cup (B_1 \setminus (P \cup A)))) = A \cup (B_1 \setminus (P \cup A)) \cup (B_2 \setminus (B_1 \cup A)) = A \cup (B_2 \setminus (P \cup A)) = \sigma(B_2)$.

In order to show naturalness of names, let D be a name-clash-free diagram and define its colimit $(C, (\mu_i)_{i \in |I|})$ as follows. C is defined by selecting from each equivalence class $X \in \mathbf{Sym}(D)/\sim_D$ a representative $r(X) \in \mathbf{Nam}(X)$ and for each index i, and each $(i, x) \in X$, we define $\mu_i(x) = r(X)$. For the particular cases of diagrams that appear in the proof already, namely those consisting of inclusions only and horizontal and vertical compositions of spans with one arrow being an inclusion, the choice of representative is determined by the respective definitions of the colimit discussed above. By name-clash-freeness, it cannot be the case that $r(X_1) = r(X_2)$ for two equivalence classes X_1, X_2. Thus, we have in C one distinct element for each equivalence class in $\mathbf{Sym}(D)/\sim_D$ and thus we have indeed selected a colimit for D.

Finally, for an arbitrary non name-clash-free diagram, select an arbitrary colimit, ensuring completeness. □

Theorem 2. $\mathbb{S}et$ does not have a selection that has total pushouts, pushout-stable inclusions and names, and coherent pushouts.

Proof. Assume that such a selection **sel** of pushouts were given. Consider the sequence of two selected pushouts (existing by total pushouts)

$$\begin{tikzcd}
\{a\} \arrow[r, hook] \arrow[d, "a\mapsto b"] & \{a,b\} \arrow[r, hook] \arrow[d, "{a\mapsto b, b\mapsto x}"] & \{a,b,x\} \arrow[d, "{a\mapsto b, b\mapsto x, x\mapsto y}"] \\
\{b\} \arrow[r, hook] & \{b,x\} \arrow[r, hook] & \{b,x,y\}
\end{tikzcd}$$

where the presence of b in the first pushout object and of b and x in the second pushout objects follows from pushout-stable inclusions. Moreover, x and y are determined by **sel**, but from the pushout property we can infer $b \neq x \neq y$. Now consider the selected pushout (again existing by total pushouts)

$$\begin{tikzcd}
\{a\} \arrow[r, hook] \arrow[d, "a\mapsto b"] & \{a,b,x\} \arrow[d, "{a\mapsto b, b\mapsto z, x\mapsto x}"] \\
\{b\} \arrow[r, hook] & \{b,z,x\}
\end{tikzcd}$$

Note that x is determined by **sel** in the previous diagram. Again, the presence of b in the pushout object follows from pushout-stable inclusions. Furthermore, the presence of x in the pushout object and the mapping $x \mapsto x$ in the pushout inclusion follows from the pushout-stable names. Moreover, z is determined by *sel*, but from the pushout property we can infer $b \neq z \neq x$.

Altogether, in the first diagram, the rightmost pushout injection maps $b \mapsto x, x \mapsto y$, while in the second diagram, it maps $b \mapsto z, x \mapsto x$. Since $z \neq x \neq y$, the maps differ. Hence, coherence does not hold. □

Proposition 5. The selection constructed in Thm. 1 can be modified to have majority-origin natural names.

Proof. Proceed as in the proof of Thm. 1, but for the "other colimits" of name-clash-free diagrams, use the majority-origin principle. If that does not determine a representative, select one of the candidates randomly. □

Theorem 3. Let $(\mathbf{C}_j)_{j\in J}$ be a family of inclusive categories with symbols and assume selections of colimits sel_j that have the properties in Thm. 1 or

Prop. 5. Then the product $\Pi_{j\in J}C_j$ can be canonically turned into a inclusive category with symbols that also has a selection of colimits **sel** with the same properties.

Proof. The product becomes an inclusive category by using tuples of inclusions as the inclusion morphisms. The symbol functors are lifted to the product by defining

$$|\_| = \biguplus_{i\in J} |\pi_j(\_)|$$

Let $D : \mathbf{I} \to \Pi_{j\in J}C_j$ be a diagram. We can define a selection of colimits by taking $\mathbf{sel}(D)$ to be the component-wise combination of $\mathbf{sel}_j(\pi_j(D))$ where π_j are the projections. It is easy to show that **sel** has the desired properties whenever the $\mathbf{sel}_j$ have them, based on the fact inclusions and colimits in products of inclusive categories are defined component-wise. ☐

Theorem 4. Let $B : \mathbf{Ind}^{op} \to \mathbb{C}at$ be an indexed inclusive category (where **Ind** is inclusive itself) such that

- B is *locally reversible*, i.e. for each $u : i \to j$ in **Ind**, $B_u : B_j \to B_i$ has a selected left adjoint $F_u : B_i \to B_j$ (note that we do not require coherence of the F_u),
- **Ind** has a selection of colimits $sel_{\mathbf{Ind}}$,
- each category B_i has a selection of colimits $\mathbf{sel}^i$, for $i \in |\mathbf{Ind}|$.

Then the Grothendieck category $B^{\#}$ is itself an inclusive category.[10]

Proof. We prove that $B^{\#}$ is inclusive. Recall that morphisms in $B^{\#}$ have form $(i, A_i) \xrightarrow{(u,\sigma)} (j, A_j)$, where $u : i \to j \in \mathbf{Ind}$ and $\sigma : A_i \to B_u(A_j) \in B_i$. Now (u, σ) is an inclusion if both u and σ are. The least element is the pair consisting of the least element $\emptyset_{\mathbf{Ind}}$ of **Ind** and the least element of $B_{\emptyset_{\mathbf{Ind}}}$. Given two objects of $B^{\#}$, (i, A_i) and (j, A_j), their union is $(i \cup j, F_{\iota_{i\subseteq i\cup j}}(A_i) \cup F_{\iota_{j\subseteq i\cup j}}(A_j))$. Given a class $\{(j, A_j)\}_{j\in J}$ of objects in $B^{\#}$, their product is $(\bigcap J, \bigcap B_{\iota_{\bigcap J\subseteq j}} A_j)$. Moreover, we have that a union-intersection square in $B^{\#}$ is a pushout. ☐

Theorem 6. Under the assumptions of Thm. 4 and Thm. 5, extended by:

- F_u preserves inclusions, and moreover,
- the unit and counit of the adjunction are inclusions.

If **Ind** and each B_i have colimit selections enjoying the properties of Thm. 1, then so does $B^{\#}$.

Proof. From the new assumptions, we can infer that

$$B_u,\ F_u,\ \_^{\#},\ \_^{\flat} \text{ and } Lift_v \text{ preserve inclusions.} \tag{1}$$

$$\text{The } \tau_{u,v} \text{ are identities, i.e. the } F_u \text{ are coherent.} \tag{2}$$

[10] Note that this construction extends to institutions, yielding Grothendieck institutions, see [5].

We can apply Theorem 2 of [23] to obtain that $B^{\#}$ has colimits. For obtaining selected colimits in $B^{\#}$, the proof from [23], which splits colimits into coproducts and coequalisers, needs to be replaced by a direct proof for all colimits. We recall two preparatory lemmas from [23]:

Lemma 1. *Given index morphisms $u : i \to j$ and $v : j \to k$, there is an isomorphism $\tau_{u,v} : F_{u;v} \to F_u; F_v$.*

Lemma 2. *Given an index morphism $v : i \to j$, any morphism $(u, \sigma) : (k, A) \to (i, B) \in B^{\#}$ can be lifted along v to a morphism in B_j:*

$$Lift_v(u, \sigma) = \tau_{u,v}; F_v(\sigma^{\#}) : F_{u;v}(A) \to F_v(B)^{11}$$

We are now prepared to compute selected colimits in $B^{\#}$. Given a diagram $D : \mathbf{I} \to B^{\#}$, let $(c, (\mu_i)_{i\in|\mathbf{I}})$ be the selected colimit of $D; \pi_1$, where π_1 is the projection to **Ind** (and π_2 the projection to the second component). Define a diagram $D' : \mathbf{I} \to B_c$ by $D'(i) = F_{\mu_i}(\pi_2(D(i)))$ and $D'(m : i \to j) = Lift_{\pi_1(D(j))}(D(m))$. Let $(C, (\nu_i)_{i\in|\mathbf{I}})$ be the selected colimit of D' in B_c. Then $((c, C), (\mu_i, \nu_i^\flat)_{i\in|\mathbf{I}})$ (where $\nu_i^\flat : \pi_2(D(i)) \to B_{\mu_i}(C)$ is adjoint to $\nu_i : F_{\mu_i}(\pi_2(D(i))) \to C$) is the selected colimit in $B^{\#}$.

We now prove the properties of the colimit selection:

Natural names : follows immediately from the assumption that the diagram $D : \mathbf{I} \to B^{\#}$ is name-clash-free and the construction of the colimit in $B^{\#}$.

Pushout-stable inclusions: Using the component-wise construction of colimits and (1).

Coherent pushouts: we treat the vertical case only. Consider the sequence of pushouts

$$\begin{tikzcd}
(p,P) \arrow[r, hook] \arrow[d, "{(u_1,\sigma_1)}"'] & (b,B) \arrow[d] \\
(a,A) \arrow[r, hook, "{(\iota,_)}"] \arrow[d, "{(u_2,\sigma_2)}"'] & \bullet \arrow[d] \\
(a',A)' \arrow[r, hook, "{(\kappa,_)}"] & \bullet
\end{tikzcd}$$

At the index level, we have

$$\begin{tikzcd}
p \arrow[r, hook] \arrow[d, "u_1"'] & b \arrow[d, "u_1^b"'] \arrow[ddr, "(u_1;u_2)^b"] & \\
a \arrow[r, hook] \arrow[d, "u_2"'] & u_1(b) \arrow[d, "u_2^{u_1(b)}"'] & \\
a' \arrow[r, hook] & u_2(u_1(b)) \arrow[r, equal] & (u_1;u_2)(b)
\end{tikzcd}$$

11 $\sigma : A \to B_u(B)$, hence its adjoint is $\sigma^{\#} : F_u(A) \to B$.

At the level of the individual fibres, the two consecutive pushouts are constructed as:

$$
\begin{tikzcd}
F_{u_1;\iota}P \arrow[r, hook] \arrow[d, "\sigma_1"'] & F_{u_1^b}B \arrow[d, "(\sigma_1^B)^\#"'] \\
F_\iota A \arrow[r, hook] & \sigma_1(B)
\end{tikzcd}
$$

$$
\begin{tikzcd}
F_{u_2;\kappa}A \arrow[r, hook] \arrow[d] & F_{u_2^{u_1(b)}}\sigma_1(B) \arrow[d, "(\sigma_2^{\sigma_1(B)})^\#"'] \\
F_\kappa A' \arrow[r, hook] & \sigma_2(\sigma_1(B))
\end{tikzcd}
$$

When applying $F_{u_2^{u_1(b)}}$ to the first diagram, by (2), both diagrams can be pasted together. The left and upper legs of this composition are identical to those of the outer pushout diagram for obtaining $((\sigma_1;\sigma_2)^B)^\#$. Then apply coherence for $B_{u_2(u_1(B))}$.

Pushout-stable names: Given a span $B \xhookleftarrow{(\iota,\_)} P \overset{(u,\sigma)}{\to} A$ in $B^\#$, natural names for pushouts give us in **Ind**:

$$
\begin{tikzcd}
p \arrow[r, hook, "\iota"'] \arrow[d, "u"] & b \arrow[d, "u^b"] & b \setminus p \arrow[l, hook', "\kappa"] \arrow[d, equal] \\
a \arrow[r, hook, "\lambda"'] & u(b) & u(b) \setminus a \arrow[l, hook', "\xi"]
\end{tikzcd}
$$

and in $B_{u(b)}$:

$$
\begin{tikzcd}
F_{u;\lambda}P \arrow[r, hook] \arrow[d, "\sigma^\#"] & F_{u^b}B \arrow[d, "(\sigma^B)^\#"] & F_{u^b}B \setminus F_{u;\lambda}P \arrow[l, hook'] \arrow[d, equal] \\
F_\lambda A \arrow[r, hook] & \sigma(B) & \sigma(B) \setminus F_\lambda A \arrow[l, hook']
\end{tikzcd}
$$

From this, we can obtain natural names for pushouts in $B^\#$.
Total pushouts: is implied by completeness.
Interchange: follows similar to natural names.
Completeness: follows from that for **Ind** and B_i.

□