

VAN KAMPEN COLIMITS AND PATH UNIQUENESS *

HARALD KÖNIG AND UWE WOLTER

Department of Informatics, University of Applied Sciences FHDW Hannover, Freundallee 15, 30173 Hannover, Germany
e-mail address: harald.koenig@fhdw.de

Department of Informatics, University of Bergen, P.O.Box 7803, 5020 Bergen, Norway
e-mail address: Uwe.Wolter@uib.no

ABSTRACT. Fibred semantics is the foundation of the model-instance pattern of software engineering. Software models can often be formalized as objects of *presheaf topoi*, i.e., categories of objects that can be represented as algebras as well as coalgebras, e.g., the category of directed graphs. Multimodeling requires to construct *colimits* of models, decomposition is given by *pullback*. Compositionality requires an exact interplay of these operations, i.e., diagrams must enjoy the *Van Kampen* property. However, checking the validity of the Van Kampen property algorithmically based on its definition is often impossible.

In this paper we state a necessary and sufficient yet efficiently checkable condition for the Van Kampen property to hold in presheaf topoi. It is based on a uniqueness property of path-like structures within the defining congruence classes that make up the colimiting cocone of the models. We thus add to the statement "Being Van Kampen is a Universal Property" by Heindel and Sobociński the fact that the Van Kampen property reveals a presheaf-based structural *uniqueness* feature.

1. INTRODUCTION

A presheaf topos is a category, that is based on an algebraic signature with unary operation symbols. Presheaves can also be considered as intersection of algebras and coalgebras [10]. Van Kampen Colimits are a generalization of Van Kampen squares [24]. In [29] we gave a necessary and sufficient condition for a pushout to be a Van Kampen square in a presheaf topos. In the present paper a corresponding criterion is given for all colimiting cocones.

Key words and phrases: Van Kampen Cocone, Presheaf Topos, Fibred Semantics.

* This paper is an extended version of the CALCO '17 paper "Being Van Kampen is a Uniqueness Property in Presheaf Topoi" [13].

1.1. Motivation. Software engineering and especially model-driven software development requires the decomposition of large models into smaller components, i.e., successful development of large applications requires system design fragmentation. Vice versa, a comprehensive viewpoint of a related ensemble of heterogenous software-engineering components is taken up by considering the *amalgamation* (union) of these artefacts modulo their relations amongst each other. This assembly shall not only be carried out on a syntactical level (models), but in the same way on the semantical level (instances). This interplay between assembly and disassembly shows that composition and *correct* decomposition of an instance of a model into instances of the model components always accompany each other. It can be shown that correctness, i.e., *compositionality* [4, 5] is not always guaranteed [22].

Fibred semantics adheres to the model-instance pattern, a standard viewpoint in software engineering: A model M is an object of an appropriate category \mathbb{C} , semantics is given by the comma category $\mathbb{C} \downarrow M$. In each object $\tau \in \mathbb{C} \downarrow M$, $\tau : I \rightarrow M$, I is the instance structure and τ is its typing. Amalgamation is colimit (of the arrangement of components) and decomposition is performed by taking pullbacks along the cocone morphisms of the colimit.

To wit: Compositionality means that colimit of semantics (instances) is controlled by colimit of syntax (models) such that pullback of the instance colimit retrieves the original instances. Thus compositionality is equivalent to the *Van Kampen property* [7], an abstract characteristic which determines an exactness level for the interaction of colimits and pullbacks. It is thus often necessary to check validity of this property. However, since the definition of the property comes in terms of an equivalence of categories, see Def.3.2 in the present paper, algorithmic verification based on the definition is hard even for a finite number of finite models, because the involved comma categories are infinite nevertheless.

Artefacts like UML- or ER-models are based on directed multigraphs, which in turn can be coded as a functor category $Set^{\mathbb{B}}$, where \mathbb{B} has objects E (edges) and V (vertices) and non-identical arrows $s, t : E \rightarrow V$. More general metamodels, however, use more sophisticated categories \mathbb{B} , such as E-graphs for attributed graphs [3], bipartite graphs for Petri nets [3], or more complex structures for generalized sketches [2]. Hence, $Set^{\mathbb{B}}$ with \mathbb{B} an arbitrary small category, will be the underlying category for the forthcoming investigations.

Constructing colimits in a category \mathbb{C} is an operation on *diagrams*, which are usually coded as functors from a small schema category \mathbb{I} to \mathbb{C} . In order to make our results usable for software engineering, we use the older definition for diagrams: Instead of a small category, the schema \mathbb{I} is a finite multigraph and a diagram is a graph morphism from \mathbb{I} to \mathbb{C} [19]¹. The practical construction of colimits relies on *mapping paths*, i.e., chains of pairs of elements that are mapped to each other by the morphisms in the diagram, cf. Def.3.3 in Sect.3. Thus, colimit computation can easily be carried out algorithmically, if the diagram is finite and consists of finite artefacts.

Summary: While colimit construction is easy, compositionality check (validation of the Van Kampen property) is hard. The first *contribution of the present paper* is a theorem (Theorem 3.5 in Sect.3), which states that a colimit in a presheaf topos has the Van Kampen property if and only if there are no ambiguous mapping paths between any pair of elements of the coproduct of the model artefacts. Thus the implementation of the colimit operation on the model level already provides the material for more efficient compositionality checking.

The second contribution is a practical algorithm, which efficiently verifies whether *compositionality* holds, i.e. whether the Van Kampen (henceforth often abbreviated "VK")

¹ More precisely to the underlying graph of \mathbb{C} , see Sect.2

property is satisfied. We provide a technique which combines (1) a more efficient colimit computation and (2) a simultaneous check of the VK-property.

The paper is organized as follows: Sect. 2 introduces notation and background information, Sect. 3 presents the main theorem and applies it to a Software Engineering problem. Sect. 4 sketches the proof idea and thus provides insight in the corresponding underlying foundations: We use a former result, in which a necessary and sufficient criterion is given for pushouts [29]. This result is translated to coequalizers and, finally, lifted to colimits of arbitrary diagrams. Sect. 5 exemplifies the limitation of Theorem 3.5: It turns out that it is crucial to claim that the underlying category is a presheaf topos and the examples show that this is the best one can achieve. The announced practical guidelines are contained in Sect. 6. Sect. 7 concludes with a short discussion of future research directions.

Proofs in the present paper are only sketched. A reader who is interested in the corresponding detailed proofs is referred to the technical report [12].

1.2. Related Work. The Van Kampen property has its origin in algebraic topology: Topological spaces X can be investigated by a covering family of X which are related by their inclusions. Topological properties are expressed with the help of the fundamental groupoid. The Van Kampen Theorem [20] states that the colimit of the fundamental groupoids of all covering spaces is the fundamental groupoid of X , thus inferring global properties from local ones. The original idea was stated by Seifert [23] for pushouts and was further elaborated by Van Kampen [26].

Inferring global properties from local ones is the heart of sheaf theory [18]. The fibred view on sheaves is discussed in [27]. The application of Van Kampen's ideas to graphical modeling and to Software Engineering was invented in [14, 24] and then further detailed in [3] for the theory of Graph Transformations. That extensive categories and especially topoi are a reasonable playground for these theories is shown in [1, 15].

Amalgamation is a requirement for a collection of artefacts in computer science [4, 5] which has been connected to the Van Kampen property in [29]. The same property is called *exactness* in institution theory [22]. Being "non-exact" seems to be a typical deficit of (1-, 2-, 3-...) categories, because Lurie shows that Higher Van Kampen theorems hold in greater generality in ∞ -topoi [17].

The importance of finding a feasible condition to check the Van Kampen property was caused by investigations of new methods in Graph Transformations [11, 16]. That the Van Kampen property can be characterized as a bicolimit in a comprising span bicategory [7] is a fundamental statement. Moreover, the Van Kampen property has been investigated in more special contexts [9] and can also be described with the help of weak 2-limits in CAT (<https://ncatlab.org/nlab/show/van+Kampen+colimit>). However, all these characterisations can hardly be applied in practice.

The present paper is an extended version of the CALCO '17 paper "Being Van Kampen is a Uniqueness Property in Presheaf Topoi". In addition to the conference version, we added examples showing the limitations of our results in Sect. 5 and we bridge the gap to concretely applicable algorithms (in Sect. 6) of the theoretical results of the paper.

2. PRELIMINARIES

This chapter recapitulates the most important notation for the following elaboration. For any category \mathbb{C} , $X \in \mathbb{C}$ means that X is contained in the collection of objects in \mathbb{C} . A

diagram in \mathbb{C} is based on a directed multigraph \mathbb{I} , the schema for the diagram. We write \mathbb{I}_0 and \mathbb{I}_1 for the sets of vertices and edges of \mathbb{I} . Formally, a diagram $\mathcal{D} : \mathbb{I} \rightarrow \mathcal{U}(\mathbb{C})$ is a graph morphism where \mathcal{U} denotes the forgetful functor assigning to each category its underlying graph. For convenience reasons, however, the forgetful functor will be omitted, i.e., diagrams will be denoted $\mathcal{D} : \mathbb{I} \rightarrow \mathbb{C}$. This definition is used instead of the one, where \mathbb{I} is a schema *category* rather than a graph, because it will turn out, that the results in this paper can easier be stated. The notions of (co-)cones and (co-)limits is the same modulo the adjunction $\mathcal{F} \dashv \mathcal{U}$ where $\mathcal{F} : \text{GRAPHS} \rightarrow \text{CAT}$ assigns to any graph its freely generated category, see [19], III, 4 for more details. Another advantage of this definition occurs in software engineering: Although the schema graph is finite, $\mathcal{F}(\mathbb{I})$ may have infinitely many arrows.

Vertices of \mathbb{I} play the role of indices for diagram objects, hence, we use letters i, j, \dots for vertices. Edges of \mathbb{I} will be depicted $i \xrightarrow{d} j$ and we write $i = s(d)$, $j = t(d)$ (source and target of d). Images of edges under a diagram $\mathcal{D} : \mathbb{I} \rightarrow \mathbb{C}$ will be denoted $\mathcal{D}_i \xrightarrow{\mathcal{D}_d} \mathcal{D}_j$ (slightly deviating from the usual notation $\mathcal{D}(i), \mathcal{D}(d)$, etc).

Let $\mathcal{E}, \mathcal{D} : \mathbb{I} \rightarrow \mathbb{C}$ be two diagrams, then a family

$$\tau = (\tau_i : \mathcal{E}_i \rightarrow \mathcal{D}_i)_{i \in \mathbb{I}_0}$$

of \mathbb{C} -morphisms with $\tau_j \circ \mathcal{E}_d = \mathcal{D}_d \circ \tau_i$ for all edges $i \xrightarrow{d} j$ in \mathbb{I}_1 will be called a *natural transformation* between the diagrams and will be denoted in the usual way $\tau : \mathcal{E} \Rightarrow \mathcal{D}$. For any $S \in \mathbb{C}$, $\Delta S : \mathbb{I} \rightarrow \mathbb{C}$ denotes the constant diagram, which sends each edge of \mathbb{I} to id_S . S (as \mathbb{C} -object) and ΔS (as diagram) will be used synonymously. Diagrams together with natural transformations constitute the category $\mathbb{C}^{\mathbb{I}}$. Note that $\Delta : \mathbb{C} \rightarrow \mathbb{C}^{\mathbb{I}}$ is itself a functor, assigning to each object of \mathbb{C} its constant diagram and to an arrow $f : A \rightarrow B$ the "constant" natural transformation $(f)_{i \in \mathbb{I}_0}$.

We assume all categories under consideration to have colimits. The coproduct cocone of a family $(\mathcal{D}_i)_{i \in I}$ of \mathbb{C} -objects will be denoted

$$(\mathcal{D}_i \xrightarrow{\subseteq_i} \coprod_{j \in I} \mathcal{D}_j)_{i \in I}.$$

The morphisms \subseteq_i are called coproduct injections. For a family of arrows $(f_i : \mathcal{D}_i \rightarrow A)_{i \in I}$ we write $\vec{f} : \coprod_{i \in I} \mathcal{D}_i \rightarrow A$ for the resulting unique mediating arrow.

We assume all categories under consideration to have pullbacks. In the sequel, we will work with *chosen pullbacks*, i.e., for each pair of \mathbb{C} -arrows $B \xrightarrow{h} A \xleftarrow{k} X$ a choice

$$\begin{array}{ccc} Y & \xrightarrow{h'} & X \\ h^*(k) \downarrow & & \downarrow k \\ B & \xrightarrow{h} & A \end{array}$$

of pullback span $(h^*(k), h')$ is determined once and for all. For all $h : B \rightarrow A$, $h^*(id_A)$ shall be chosen to be id_B . Whenever we deviate from these choices, this will be emphasized. It is well-known [6] that for fixed $h : B \rightarrow A$ chosen pullbacks along h give rise to a (pullback) functor $h^* : \mathbb{C} \downarrow A \rightarrow \mathbb{C} \downarrow B$ between comma categories. Pullbacks can be composed, i.e., if $C \xrightarrow{h_2} B \xrightarrow{h_1} A$, then $h_2^* \circ h_1^*$ yields a pullback along $h_1 \circ h_2$, and decomposed, i.e., if $h_1^*(k)$ and $(h_1 \circ h_2)^*(k)$ are computed, the resulting universal arrow from the latter into the former

pullback yields a pullback of h_2 and $h_1^*(k)$. Note, that in both cases the automatically appearing pullbacks need not be chosen.

The underlying category for all further considerations is a category of *presheaves*, i.e., the category $\mathbb{G} := \text{Set}^{\mathbb{B}}$ (with \mathbb{B} a small (base) category, *Set* the category of sets and mappings) of covariant functors from \mathbb{B} to *Set* together with natural transformations between them². We will also use the term "sort" for the objects in \mathbb{B} and the term "operation (symbol)" for the morphisms in \mathbb{B} . It is folklore that \mathbb{G} has all colimits and all pullbacks, which are computed sortwise, resp. \mathbb{G} is a topos, i.e. a category with finite limits and colimits, which has exponents and where the subobject functor is representable [6]. \mathbb{G} will thus also be called a *presheaf topos*. E.g., the category of multigraphs is a presheaf topos with $\mathbb{B} = (E \xrightarrow{s} V)$ (plus identities). The simplest presheaf topos is *Set* ($\mathbb{B} = 1$, the one-object-one-morphism category).

In this paper, we will make frequent use of (sortwise) coproducts, i.e., disjoint unions of sets. In order to make argumentations simpler, we will assume that for each $X \in \mathbb{B}$ the artefacts $(\mathcal{D}_i(X))_{i \in \mathbb{I}_0}$ are a priori disjoint, i.e., the coproduct is obtained by simple union.

An important property of presheaf topoi is (infinite) *extensivity*, i.e., the functor

$$\coprod : \prod_{i \in I} \mathbb{G} \downarrow D_i \rightarrow \mathbb{G} \downarrow \coprod_{i \in I} D_i \tag{2.1}$$

assigning to each object $(f_i : A_i \rightarrow D_i)_{i \in I}$ in $\prod_{i \in I} \mathbb{G} \downarrow D_i$ the object $\coprod_{i \in I} f_i : \coprod_{i \in I} A_i \rightarrow \coprod_{i \in I} D_i$ in $\mathbb{G} \downarrow \coprod_{i \in I} D_i$, is an equivalence of categories for each index set I and each I -indexed family $(D_i)_{i \in I}$ of objects in \mathbb{G} . Its "inverse" arises from constructing pullbacks along coproduct injections. From these facts one derives the stability of coproducts under pullbacks, i.e., if

$$\begin{array}{ccc} A_i & \xrightarrow{a_i} & A \\ f_i \downarrow & & \downarrow g \\ M_i & \xrightarrow{g_i} & M \end{array} \qquad \begin{array}{ccc} \coprod_{i \in I} A_i & \xrightarrow{\bar{a}} & A \\ \coprod_{i \in I} f_i \downarrow & & \downarrow g \\ \coprod_{i \in I} M_i & \xrightarrow{\bar{g}} & M \end{array} \tag{2.2}$$

are commutative diagrams, then the squares on the left-hand side are pullbacks for all $i \in I$, if and only if the square on the right-hand side is a pullback [6]. In general topoi, all these statements still hold for finite index sets I (finite extensivity).

3. AN EQUIVALENT CONDITION FOR THE VAN KAMPEN PROPERTY

In this chapter we introduce the Van Kampen property and state the main result of this paper, a necessary and sufficient condition for the Van Kampen property to hold in $\mathbb{G} = \text{Set}^{\mathbb{B}}$.

3.1. Van Kampen Colimits. A commutative cocone out of a diagram $\mathcal{D} : \mathbb{I} \rightarrow \mathbb{G}$ is a natural transformation

$$\kappa : \mathcal{D} \Rightarrow \Delta S. \tag{3.1}$$

² Normally presheaves are categories $\text{Set}^{\mathbb{B}^{op}}$, i.e., contravariant *Set*-valued functors. But we prefer the slightly deviating definition, because we found the contravariant version counterintuitive for our work. Clearly, it is easy to switch to the contravariant setting, if one inverts all arrows of \mathbb{B} .

For fixed $i \xrightarrow{d} j$ of \mathbb{I}_1 , pulling back a \mathbb{G} -arrow $K \xrightarrow{\sigma} S$ along κ_i and κ_j yields

$$\begin{array}{ccccc}
 & & \xrightarrow{\kappa'_i} & & \\
 \mathcal{E}_i & \xrightarrow{\mathcal{E}_d} & \mathcal{E}_j & \xrightarrow{\kappa'_j} & K \\
 \downarrow \kappa_i^*(\sigma) & & \downarrow \kappa_j^*(\sigma) & & \downarrow \sigma \\
 \mathcal{D}_i & \xrightarrow{\mathcal{D}_d} & \mathcal{D}_j & \xrightarrow{\kappa_j} & S \\
 & & \xrightarrow{\kappa_i} & &
 \end{array} \tag{3.2}$$

where the right and the outer rectangles are chosen pullbacks, \mathcal{E}_d is the unique completion into the right pullback, and the resulting left square is a pullback by the pullback decomposition property. The left square may, however, not be a chosen one, but it results in diagram \mathcal{E} as well as natural transformation $\kappa^*(\sigma) := (\kappa_i^*(\sigma))_{i \in \mathbb{I}_0} : \mathcal{E} \Rightarrow \mathcal{D}$, whose naturality squares are pullbacks. This fact gives rise to the following definition:

Definition 3.1 (Cartesian Transformation). A natural transformation $\tau : \mathcal{E} \Rightarrow \mathcal{D} : \mathbb{I} \rightarrow \mathbb{G}$ is called *cartesian* if all naturality squares are pullbacks.

For a fixed diagram $\mathcal{D} : \mathbb{I} \rightarrow \mathbb{G}$ let $\mathbb{G}^{\mathbb{I}} \Downarrow \mathcal{D}$ be the full subcategory of $\mathbb{G}^{\mathbb{I}} \downarrow \mathcal{D}$ of *cartesian* natural transformations. Thus, by (3.2), κ^* maps objects of $\mathbb{G} \downarrow S$ to objects in $\mathbb{G}^{\mathbb{I}} \Downarrow \mathcal{D}$. Moreover, any arrow $\gamma : \sigma \rightarrow \sigma'$ of $\mathbb{G} \downarrow S$ yields a family of arrows $(\kappa_i^*(\gamma))$ (universal arrows into pullbacks) of which it can easily be shown that together they yield a cartesian natural transformation $\kappa^*(\gamma) : \kappa^*(\sigma) \rightarrow \kappa^*(\sigma')$. Thus κ^* becomes a functor

$$\kappa^* : \mathbb{G} \downarrow S \rightarrow \mathbb{G}^{\mathbb{I}} \Downarrow \mathcal{D}. \tag{3.3}$$

Definition 3.2 (Van Kampen Cocone, [7]). Let $\mathcal{D} : \mathbb{I} \rightarrow \mathbb{G}$ be a diagram and $\kappa : \mathcal{D} \Rightarrow \Delta S$ be a commutative cocone. Then κ has the *Van Kampen (VK) Property* (" κ is VK") if functor κ^* is an equivalence of categories.

As usual, a *colimit* (or *colimiting cocone*) is a universal cocone $\kappa : \mathcal{D} \Rightarrow \Delta S$, i.e., for each $T \in \mathbb{G}$ and commutative cocone $\rho : \mathcal{D} \Rightarrow \Delta T$, there is a unique \mathbb{G} -morphism $S \xrightarrow{u} T$ such that $\Delta u \circ \kappa = \rho$, i.e., $u \circ \kappa_i = \rho_i$ for all $i \in \mathbb{I}_0$. S is called the *colimit object*.

κ^* has a left-adjoint $\kappa_* : \mathbb{G}^{\mathbb{I}} \Downarrow \mathcal{D} \rightarrow \mathbb{G} \downarrow S$ which assigns to a cartesian natural transformation $\tau : \mathcal{E} \Rightarrow \mathcal{D}$ the unique arrow to S out of the colimit object of the colimiting cocone of \mathcal{E} [24]. I.e., κ_* is the (pseudo-)inverse of κ^* , if the VK property holds. In this case, unit and counit of the adjunction are isomorphisms. Note also that each VK cocone $\mathcal{D} \Rightarrow \Delta S$ is automatically a colimit (apply κ^* to $id_{\Delta S}$ and use the definition of κ_*) such that we can use the terms "Van Kampen cocone" and "Van Kampen colimit" synonymously.

Whereas the counit of this adjunction is always an isomorphism, if pullback functors have right-adjoints (and thus preserve colimits), which is true in every (presheaf) topos [6], the situation is more involved concerning the unit of the adjunction: The easiest example of the VK property arises for the empty diagram. In this case the property translates to the fact, that the *initial object* 0 is strict, i.e., each arrow $A \longrightarrow 0$ is an isomorphism. This is true in all topoi [6]. In the same way, since all presheaf topoi are extensive (cf. Sect.2), coproducts have the Van Kampen property. But the unit fails to be an isomorphism for pushouts and coequalizers: Even in *Set* there are easy examples of pushouts which violate the VK property [24]. In *adhesive categories* (and thus in all topoi [15]) pushouts are VK, if one leg is monic, by definition. Vice versa, there are also pushouts with both legs non-monic, which enjoy this property nevertheless [29]. Astonishingly, coequalizers

seldom are VK: Consider the shape graph $\mathbf{2} := 1 \xrightarrow[d]{d'} 2$ and the diagram $\mathcal{D} : \mathbf{2} \rightarrow \mathit{Set}$ with $\mathcal{D}_1 = \{*\}_1, \mathcal{D}_2 = \{*\}_2$. Clearly,

$$\mathcal{D}_1 \rightrightarrows \mathcal{D}_2 \longrightarrow \{*\} \quad (3.4)$$

is a coequalizer in Set . Then the cartesian transformation

$$\tau : (\mathcal{E}_1 := \{a, b\} \xrightarrow[k]{id} \mathcal{E}_2 := \{a, b\}) \Rightarrow \mathcal{D},$$

with k the non-identical bijection of $\{a, b\}$, is mapped to $id_{\{*\}}$ by κ_* , i.e., $\tau \notin (\kappa^* \circ \kappa_*)(\tau)$.

3.2. Equivalent Condition. As mentioned in the introduction it is important for several software engineering scenarios to find an easily checkable criterion for the Van Kampen property. The presented condition of this paper comes in terms of the mapping behavior of all morphisms \mathcal{D}_d in the diagram.

Definition 3.3 (Mapping Path). Let $\mathbb{G} = \mathit{Set}^{\mathbb{B}}$ be a presheaf topos and $\mathcal{D} : \mathbb{I} \rightarrow \mathbb{G}$ be a diagram w.r.t. shape graph \mathbb{I} . Let $\mathbb{I}_1^{op} := \{d^{op} \mid d \in \mathbb{I}_1\}$.

- A *Path Segment* of sort $X \in \mathbb{B}$ is a triple (y, δ, y') with $\delta \in \mathbb{I}_1 \cup \mathbb{I}_1^{op}$ and³

$$\begin{array}{ll} \text{If } \delta = d \in \mathbb{I}_1 & \text{then } y \in \mathcal{D}_{s(d)}(X), y' = \mathcal{D}_d(y) \in \mathcal{D}_{t(d)}(X) \\ \text{If } \delta = d^{op} \in \mathbb{I}_1^{op} & \text{then } y' \in \mathcal{D}_{s(d)}(X), y = \mathcal{D}_d(y') \in \mathcal{D}_{t(d)}(X) \end{array}$$

Two path segments (y_1, δ_1, y'_1) and (y_2, δ_2, y'_2) of sort X are equal if $y_1 = y_2$, $\delta_1 = \delta_2$, and $y'_1 = y'_2$. Moreover, two path segments are *weakly equal*, $(y_1, \delta_1, y'_1) =_w (y_2, \delta_2, y'_2)$ in symbols, if $(y_1, \delta_1, y'_1) = (y_2, \delta_2, y'_2)$ or $(y_1, \delta_1, y'_1) = (y'_2, \delta_2^{op}, y_2)$.⁴

- A *Non-empty Mapping Path in \mathcal{D}* of sort $X \in \mathbb{B}$ is a sequence

$$P = [(y_0, \delta_0, y_1), (y_1, \delta_1, y_2), (y_2, \delta_2, y_3), \dots, (y_{n-1}, \delta_{n-1}, y_n)]$$

of path segments of sort X , where any third component of a segment coincides with the first component of its successor segment⁵, and where $n \geq 1$. We say that the above path connects y_0 with y_n in \mathcal{D} .

- For each $y \in \mathcal{D}_i(X)$, where $i \in \mathbb{I}_0$ and $X \in \mathbb{B}$, we say that the *Empty Mapping Path* $[]$ of sort X connects y with itself in \mathcal{D} .
- Two paths are equal, if they have the same length and are segmentwise equal.
- A mapping path is *proper* if there are no two distinct path segments that are weakly equal.

Examples of mapping paths for graphs are depicted in Fig.1 (the complete meaning of the contents of Fig.1 will be explained in the next section): There are two paths (one along the dashed path segments, the other one along the dotted segments) both connecting vertex "Sort" with vertex "Type". Each arrow depicts a path segment with first component the arrow's source and third component its target. The middle component is annotated near the arrows, resp., their names will be explained in the next section, as well.

³ Whenever $i \xrightarrow{d} j \in \mathbb{I}_1$ and we apply a mapping in the family $((\mathcal{D}_d)_X : \mathcal{D}_i(X) \rightarrow \mathcal{D}_j(X))_{X \in \mathbb{B}}$, we write \mathcal{D}_d instead of $(\mathcal{D}_d)_X$.

⁴ $(d^{op})^{op} := d$.

⁵ By the introductory remarks on disjointness of artefacts, this means that the third component and the successor's first component are elements of the same \mathcal{D}_i .

For any $X \in \mathbb{B}$, any $i, j \in \mathbb{I}_0$ and any $z \in \mathcal{D}_i(X)$, $z' \in \mathcal{D}_j(X)$ we write $z \equiv_X z'$ ($z \equiv_X^p z'$), if there is a mapping path (proper mapping path) of sort X connecting z with z' . It is easy to see that $\equiv = \equiv^p$ and that this relation is a congruence relation on $\coprod_{i \in \mathbb{I}_0} \mathcal{D}_i$ (i.e., a family of equivalence relations $(\equiv_X)_{X \in \mathbb{B}}$ compatible with operations of \mathbb{B}), because paths can be concatenated and reversed. Moreover, it is well-known [19] that the colimiting cocone of diagram $\mathcal{D} : \mathbb{I} \rightarrow \mathbb{G}$ is given by

$$\mathcal{D} \xrightarrow{\kappa} \left(\coprod_{i \in \mathbb{I}_0} \mathcal{D}_i \right) / \equiv = \left(\coprod_{i \in \mathbb{I}_0} \mathcal{D}_i \right) / \equiv^p \quad (3.5)$$

where $\kappa_i = []_{\equiv} \circ \subseteq_i$ with $[]_{\equiv}$ the canonical morphism. In the present paper we will show that mapping paths also play a crucial role for a simpler characterization of the Van Kampen property. The following examples hint at this connection.

Example 3.4. Let $\mathbb{G} = \text{Set}$.

- (1) In (3.4) there are proper mapping paths $[]$ and $[(*_2, d^{op}, *_1), (*_1, d', *_2)]$ both connecting $*_2$ with itself.
- (2) The shape graph $1 \xleftarrow{d} 0 \xrightarrow{d'} 2$ yields pushouts. The easiest example of a non-VK pushout arises from $\mathcal{D}_0 = \{x, y\}$, $\mathcal{D}_1 = \{*_1\}$, $\mathcal{D}_2 = \{*_2\}$, cf. [24]. In this case, we obtain two different proper mapping paths $[(*_1, d^{op}, x), (x, d', *_2)]$ and $[(*_1, d^{op}, y), (y, d', *_2)]$ both connecting $*_1$ and $*_2$ in \mathcal{D} .
- (3) Let $\mathbb{I} = d \begin{array}{c} \curvearrowright \\ \bullet \end{array}$ consist of one vertex and one loop. I.e., diagrams depict endomorphisms $f : A \rightarrow A$. It is astonishing that even the colimiting cocone $\mathcal{D} \Longrightarrow \{*\}$ with $\mathcal{D}_d = id_{\{*\}}$ is not VK: Take $\mathcal{E} = (\{a, b\} \xrightarrow{k} \{a, b\})$ (with k the non-identity bijection of $\{a, b\}$), $\tau : \{a, b\} \rightarrow \{*\}$, then \mathcal{E} 's colimit is a singleton. In this example, we have two proper mapping paths $[]$ and $[(*, d, *)]$ in \mathcal{D} both connecting $*$ with itself. Note that this is just another presentation of example (3.4), since the colimit of $f : A \rightarrow A$ can be obtained by the coequalizer of $A \xrightarrow{id_A} A \xrightarrow{f} A$.

- (4) $\mathcal{D} = (\{x\} \xrightarrow[g]{f} \{y, z\})$ with $f(x) = y, g(x) = z$ has the VK property, which can be checked by elementary means based on Def. 3.2. There is exactly one proper mapping path connecting y and z , namely $[(y, f^{op}, x), (x, g, z)]$. Moreover, there is exactly one proper path connecting y with itself (namely the empty one, the hypothetical path $[(y, f^{op}, x), (x, f, y)]$ is not proper, see Def.3.3). In the same way x has only one path back to itself, namely the empty one (the hypothetical path $[(x, f, y), (y, f^{op}, x)]$ is not proper).

As suggested by these examples, *uniqueness of proper mapping paths* between two elements of the same sort X in the sets $(\mathcal{D}_i(X))_{i \in \mathbb{I}_0}$ is a crucial feature for the Van Kampen property to hold. Indeed, we will prove

Theorem 3.5 (VK is Path Uniqueness). *Let $\mathbb{G} = \text{Set}^{\mathbb{B}}$ be a presheaf topos and $\mathcal{D} : \mathbb{I} \rightarrow \mathbb{G}$ be a diagram. Let $\mathcal{D} \xrightarrow{\kappa} \Delta S$ be a colimiting cocone. The cocone is a Van Kampen cocone if and only if for all $X \in \mathbb{B}$, all $i, j \in \mathbb{I}_0$ and all $z \in \mathcal{D}_i(X)$, $z' \in \mathcal{D}_j(X)$: There are no two different proper mapping paths in \mathcal{D} connecting z and z' .*

Since, in colimit computations, all mapping paths need to be computed (see (3.5)), and – according to Theorem 3.5 – the Van Kampen property can be checked by means of mapping paths, algorithmic verification of the Van Kampen property can be carried out in the background of colimit computation. A detailed elaboration of these combined computations is carried out in Sect.6.

3.3. Application of Theorem 3.5. In order to demonstrate the benefits of the path uniqueness criterion, we consider a more substantial example than the ones in Example 3.4.

We let $\mathbb{B} = (E \xrightarrow{s} V)$ (id_E and id_V not shown), thus our base presheaf topos is $\mathbb{G} = Set^{\mathbb{B}}$, the category of directed multigraphs. In the sequel, we depict vertices as rectangles and edges are arrows pointing from its source to its target. In Fig.1 the three highlighted graphs⁶ \mathcal{D}_1 , \mathcal{D}_2 , and \mathcal{D}_3 depict meta-models for type systems:

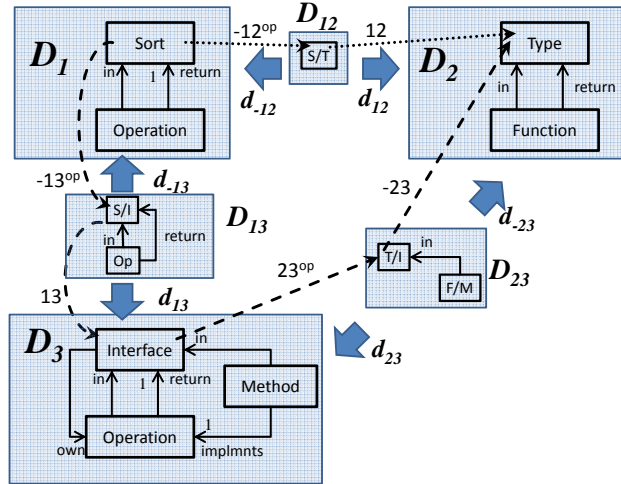


Figure 1: A diagram of metamodels and two mapping paths

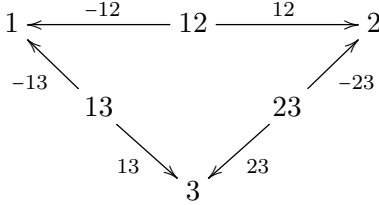
- \mathcal{D}_1 represents parts of the domain of *algebraic specifications*: Operations have an arbitrary number of sort-typed input parameters and exactly one return parameter.
- In \mathcal{D}_2 terminology of *abstract data types* is used: Functions have an arbitrary number of typed input and return parameters, resp.
- \mathcal{D}_3 is the object-oriented view: Interfaces own operations, which have inputs and one return parameter typed in interfaces, resp. Methods implement operations, their input parameters may be of specialized type.

Figure 1 represents a *multimodeling* scenario [21]. Reasoning about these collective models (the multimodel) as one artefact requires matching of different terminology of each of the model graphs: Sameness of terminology in graphs \mathcal{D}_1 and \mathcal{D}_2 is formally enabled by defining a relation on $\mathcal{D}_1 \times \mathcal{D}_2$ by means of auxiliary graph \mathcal{D}_{12} , which consists of exactly one vertex S/T ,

⁶These are not just graphs since they contain "multiplicity constraints". They can be formalized, actually, as generalized sketches, i.e., graphs with diagrammatic predicates, in the sense of [2].

$d_{-12}(S/T) = \text{Sort}$, $d_{12}(S/T) = \text{Type}$, such that span $\mathcal{D}_1 \xleftarrow{d_{-12}} \mathcal{D}_{12} \xrightarrow{d_{12}} \mathcal{D}_2$ specifies sameness of terms "Sort" and "Type" in graphs \mathcal{D}_1 , \mathcal{D}_2 and no other commonalities. In the same way span $\mathcal{D}_1 \xleftarrow{d_{-13}} \mathcal{D}_{13} \xrightarrow{d_{13}} \mathcal{D}_3$ specifies sameness of terms "Sort" and "Interface" (in \mathcal{D}_1 and \mathcal{D}_3) as well as "Operation" (in both graphs) together with the in- and return-relationships. Moreover, relation "in" of term "Method" in \mathcal{D}_3 is declared to be equal to property "in" of term "Function" in \mathcal{D}_2 via span $\mathcal{D}_3 \xleftarrow{d_{23}} \mathcal{D}_{23} \xrightarrow{d_{-23}} \mathcal{D}_2$.

We now describe a scenario, in which colimit computation of the graphs in Fig.1 and amalgamation of instances typed over these graphs is important. It is common to reason about the multimodel by imposing constraints that spread over different models. We could, e.g., claim that "The return type of a method's implemented operation (as specified in \mathcal{D}_3) has to be contained in the list of return types of the corresponding function (as specified in \mathcal{D}_2)". In order to check this inter-model constraint, it is necessary to construct the diagram's colimit. Formally, for schema graph $\mathbb{I} =$



we obtain diagram $\mathcal{D} : \mathbb{I} \rightarrow \mathbb{G}$ and construct the colimiting cocone $\mathcal{D} \xrightarrow{\kappa} \Delta S$.

Assume now that we want to check consistency of given typed instances $\tau_i : \mathcal{E}_i \rightarrow \mathcal{D}_i$, $i \in \{1, 2, 3\}$ against the above formulated constraint. For this we have to declare sameness of elements within $\mathcal{E}_1, \mathcal{E}_2$, and \mathcal{E}_3 with the help of new relating typing morphisms $\tau_k : \mathcal{E}_k \rightarrow \mathcal{D}_k$, $k \in \{12, 13, 23\}$ and spans $\mathcal{E}_1 \xleftarrow{e_{-12}} \mathcal{E}_{12} \xrightarrow{e_{12}} \mathcal{E}_2$, $\mathcal{E}_1 \xleftarrow{e_{-13}} \mathcal{E}_{13} \xrightarrow{e_{13}} \mathcal{E}_3$, and $\mathcal{E}_2 \xleftarrow{e_{-23}} \mathcal{E}_{23} \xrightarrow{e_{23}} \mathcal{E}_3$. Of course, all τ_k have to be compatible with model matching and sameness declaration within $\mathcal{E}_1, \mathcal{E}_2$, and \mathcal{E}_3 , i.e., we obtain a natural transformation $\tau : \mathcal{E} \Rightarrow \mathcal{D}$ between diagrams of type $\mathbb{I} \rightarrow \mathbb{G}$. Consistency checking is then carried out by constructing the colimit object K of \mathcal{E} and checking whether the resulting typing arrow $\sigma : K \rightarrow S$ fulfills the constraint, see [21].

Let us momentarily ignore constraint checking and just consider the relation between this amalgamated instance σ and the original component instances $(\tau_i)_{i \in \{1, 2, 3, 12, 13, 23\}}$: It is necessary to faithfully recover all τ_i from σ , otherwise we would loose information about the origin of the elements in the domain of σ . This means that we require, for all i , $\kappa_i^*(\sigma) \cong \tau_i$, i.e., the Van Kampen property for the cocone κ has to hold. However, it turns out, that the property is violated: This can be seen by considering the following instance constellation (we write $x:T$ whenever $\tau(x) = T$): Let $\mathcal{E}_1(V) = \{s: \text{Sort}, s': \text{Sort}\}$, $\mathcal{E}_1(E) = \emptyset$, $\mathcal{E}_2(V) = \{t_1: \text{Type}, t_2: \text{Type}\}$, $\mathcal{E}_2(E) = \emptyset$, and $\mathcal{E}_3(V) = \{\bar{i}: \text{Interface}, i: \text{Interface}\}$, $\mathcal{E}_3(E) = \emptyset$. One may now declare sameness of elements within $\mathcal{E}_1, \mathcal{E}_2$, and \mathcal{E}_3 as follows

$$s = t_1, s' = t_2 \text{ by span } (e_{-12}, e_{12}); s = i, s' = \bar{i} \text{ by } (e_{-13}, e_{13}); t_1 = \bar{i}, t_2 = i \text{ by } (e_{-23}, e_{23}).$$

This is established as described above, e.g., graph \mathcal{E}_{12} consists of two vertices $1: S/T$ and $2: S/T$. Graph morphisms e_{-12} maps $1: S/T \mapsto s$ and $2: S/T \mapsto s'$ whereas e_{12} maps $1: S/T \mapsto t_1$ and $2: S/T \mapsto t_2$. We omit the obvious formal definitions of the other two spans.

Unfortunately, by transitivity, this matching also yields $s = s'$, an unwanted anomaly. But in practice this effect may happen, if two modelers work separately: One modeler might define matches (e_{-12}, e_{12}) and (e_{-13}, e_{13}) and, independently and inadvertently, the second

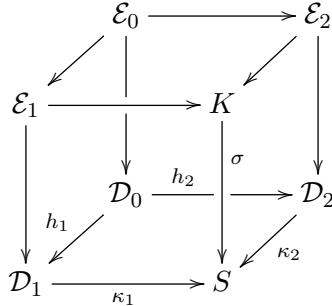
modeler defines the match (e_{-23}, e_{23}) . The inconsistent matching yields a colimit K of \mathcal{E} with one vertex only, because each sort/type/interface is connected with each other along mapping paths. Clearly, $\kappa_i^*(\sigma) \neq \tau_i$ since κ_i are monomorphisms, hence the domains of $\kappa_i^*(\sigma)$ are singleton sets, as well.

In this simple example, a small instance constellation allowed for the detection of a VK violation witness. However, it is hard to determine such witnesses in more complex examples. In these cases, Theorem 3.5 is a more reliable indicator for VK validity or violation, because we do not need to find violating instance constellations. Instead, the violation of the VK property can be detected by analysing mapping path structures of the metamodels only. In the present example, the indicator are the *two different proper mapping paths* of sort V shown in Fig.1 both connecting "Sort" and "Type" (one is depicted by dashed, the other one by dotted arrows) such that Theorem 3.5 immediately yields violation of the Van Kampen property. At least from this example we derive the slogan that *the Van Kampen property holds, if there is no redundant matching information* in \mathcal{D} . It is easy to see that the negative effect vanishes if we reduce the diagram accordingly, i.e. if we erase matching via \mathcal{D}_{12} since this information is already contained in the transitive closure of matchings \mathcal{D}_{13} and \mathcal{D}_{23} . In this way, the above mentioned modelers can indeed work independently!

4. AN OUTLINE OF THE PROOF OF THEOREM 3.5

In this section, we sketch the main steps for the proof of our main theorem. Each step is given by a Lemma for which detailed proofs can be found in the technical report [12].

4.1. Pushouts. Often, the Van Kampen property for pushouts is formulated as follows: A pushout of a diagram $\mathcal{D}_1 \xleftarrow{h_1} \mathcal{D}_0 \xrightarrow{h_2} \mathcal{D}_2$ is said to have the Van Kampen property if for any commutative cube



with this pushout in the bottom and back faces pullbacks, the front faces are pullbacks if and only if the top face is a pushout. In [29] we already stated a characterization of the Van Kampen property for pushouts based on this definition. It comes in terms of cyclic mapping structures within \mathcal{D}_0 :

Definition 4.1 (Domain Cycle, [16]). Consider a span $\mathcal{D}_1 \xleftarrow{h_1} \mathcal{D}_0 \xrightarrow{h_2} \mathcal{D}_2$ in $\mathbb{G} = \text{Set}^{\mathbb{B}}$. For $X \in \mathbb{B}$ we call a sequence $[x_0, x_1, \dots, x_{2k+1}]$ of elements of $\mathcal{D}_0(X)$ a *domain cycle* (for the span (h_1, h_2)) (of sort X), if $k \in \mathbb{N}$ and the following conditions hold:

- (1) $\forall j \in \{0, 1, \dots, 2k+1\} : x_j \neq x_{j+1}$
- (2) $\forall i \in \{0, \dots, k\} : h_1(x_{2i}) = h_1(x_{2i+1})$
- (3) $\forall i \in \{0, \dots, k\} : h_2(x_{2i+1}) = h_2(x_{2i+2})$

where $2k + 2 := 0$. A domain cycle is *proper* if $x_i \neq x_j$ for all $0 \leq i < j \leq 2k + 1$.

The main outcome of [29] is the following fact:

Lemma 4.2 (Condition for VK Pushouts). *A pushout*

$$\begin{array}{ccc} \mathcal{D}_0 & \xrightarrow{h_2} & \mathcal{D}_2 \\ h_1 \downarrow & & \downarrow \kappa_2 \\ \mathcal{D}_1 & \xrightarrow{\kappa_1} & S \end{array}$$

in $\mathbb{G} = \text{Set}^{\mathbb{B}}$ is a Van Kampen cocone iff there is no proper domain cycle for (h_1, h_2) . \square

This result can be proven by means of elementary set-based arguments [16], but also by investigating forgetful functors between categories of descent data [8] for general topoi [29].

It is easy to see that the above definition for pushouts is an instance of the general definition of Van Kampen colimits in Def. 3.2:

- If the front faces are pullbacks, then the back faces are the result of applying κ^* . Then the counit $\varepsilon : \kappa_* \circ \kappa^* \Rightarrow Id$ of adjunction is an isomorphism if and only if the cube's top face is already a pushout.
- If the top face is a pushout, then (up to isomorphism) σ is the result of applying κ_* . Then the unit $\eta : Id \Rightarrow \kappa^* \circ \kappa_*$ is an isomorphism if and only if $\kappa^*(\sigma)$ produces the original cube up to isomorphism, i.e., the original front faces are pullbacks.

Hence, the two implications "Front face pullbacks iff top face pushout" actually reflect the two statements "The counit is an isomorphism" and "The unit is an isomorphism".

Thus Lemma 4.2 is a good starting point for the proof of Theorem 3.5: We first transfer this knowledge to special mapping paths in coequalizer diagrams (Sect.4.2) and then from there to mapping paths in arbitrary colimits (Sect.4.3).

4.2. From Pushouts to Coequalizers. The transfer from pushouts to coequalizers is accomplished in two steps. The first step connects the VK property for coequalizers and pushouts:

Lemma 4.3. *Let \mathbb{G} be a general topos and $B \rightrightarrows_f^g D$ be two arrows in \mathbb{G} . Let two arrows $\kappa_D : D \rightarrow S$ and $\kappa_B : B \rightarrow S$ be given such that the diagrams*

$$\begin{array}{ccc} B & \xrightarrow{g} & D \xrightarrow{\kappa_D} S \\ & \searrow f & \nearrow \kappa_B \\ & & \end{array} \quad \begin{array}{ccc} B + B & \xrightarrow{[f,g]} & D \\ [id,id] \downarrow & & \downarrow \kappa_D \\ B & \xrightarrow{\kappa_B} & S \end{array}$$

are commutative, resp.

- (1) *The left diagram is a coequalizer if and only if the right diagram is a pushout.*
- (2) *The left diagram is a VK cocone if and only if the right diagram is.*

Proof. 1 is well-known [19]. 2 is proven by means of Def.3.2, where the transfer is possible, because topoi are (finitely) extensive, cf. Sect.2, and especially because of property (2.2). \square

The second step establishes a connection between domain cycles and mapping paths. It comes in terms of *disjoint* mapping paths, i.e., paths P_1 and P_2 for which none of the path segments in P_1 is weakly equal⁷ to a path segment in P_2 . The proof is rather technical and will be omitted, see [12], Lemma 14.

Lemma 4.4 (Domain Cycles vs. Mapping Paths). *Let $\mathbb{G} = \text{Set}^{\mathbb{B}}$, f and g as in Lemma 4.3, and $X \in \mathbb{B}$. There is a proper domain cycle of sort X for $B \xleftarrow{[id, id]} B + B \xrightarrow{[f, g]} D$, if and only if there are $z, z' \in D(X)$ and two disjoint proper mapping paths connecting z and z' . \square*

Lemmas 4.2, 4.3, and 4.4 yield

Corollary 4.5 (Condition for VK Coequalizers). *Let $\mathbb{G} = \text{Set}^{\mathbb{B}}$, let $\mathfrak{2}$ be the schema graph $1 \xrightleftharpoons[d]{d'} \mathfrak{2}$, and $\bar{\mathcal{D}} : \mathfrak{2} \rightarrow \mathbb{G}$. The coequalizer diagram*

$$\begin{array}{ccc} \bar{\mathcal{D}}_1 & \begin{array}{c} \xrightarrow{\bar{\mathcal{D}}_{d'}} \\ \xrightarrow{\bar{\mathcal{D}}_d} \\ \xrightarrow{\kappa_1} \end{array} & \bar{\mathcal{D}}_2 \xrightarrow{\kappa_2} S \end{array}$$

has the Van Kampen property, if and only if for all $X \in \mathbb{B}$ and all $z, z' \in \bar{\mathcal{D}}_2(X)$: There are no two disjoint proper mapping paths of sort X in $\bar{\mathcal{D}}$ connecting z and z' . \square

Recall the already made observations in Example 3.4, 1. and 4., which confirm this statement.

4.3. From Coequalizers to Colimits. It is well-known [19], that the colimit of $\mathcal{D} : \mathbb{I} \rightarrow \mathbb{G}$ can be computed by constructing the coequalizer of

$$\coprod_{d \in \mathbb{I}_1} \mathcal{D}_{s(d)} \xrightarrow[\bar{\mathcal{D}}_d]{\bar{i}d} \coprod_{j \in \mathbb{I}_0} \mathcal{D}_j, \quad (4.1)$$

where $\bar{\mathcal{D}}_d$ and $\bar{i}d$ are mediators out of the involved coproducts:

$$\begin{array}{ccc} \coprod_{d \in \mathbb{I}_1} \mathcal{D}_{s(d)} & \xrightarrow{\bar{\mathcal{D}}_d} & \coprod_{j \in \mathbb{I}_0} \mathcal{D}_j & \quad & \coprod_{d \in \mathbb{I}_1} \mathcal{D}_{s(d)} & \xrightarrow{\bar{i}d} & \coprod_{i \in \mathbb{I}_0} \mathcal{D}_i & (4.2) \\ \subseteq_{i,d} \uparrow & & \uparrow \subseteq_j & & \subseteq_{i,d} \uparrow & & \uparrow \subseteq_i & \\ \mathcal{D}_i & \xrightarrow{\mathcal{D}_d} & \mathcal{D}_j & & \mathcal{D}_i & \xlongequal{\quad} & \mathcal{D}_i & \end{array}$$

(for all edges $i \xrightarrow{d} j$ in \mathbb{I}_1).⁸

Let $\bar{\mathcal{D}} : \mathfrak{2} \rightarrow \mathbb{G}$ be the functor mapping $\mathfrak{2}$ to the objects and arrows in (4.1), where schema graph $\mathfrak{2}$ is given as before (cf. e.g. Cor. 4.5). Then a technical analysis shows that we can combine mapping paths of \mathcal{D} with special mapping paths of $\bar{\mathcal{D}}$ (again, we omit the proof and refer to [12]):

Lemma 4.6. *Let $\mathbb{G} = \text{Set}^{\mathbb{B}}$ and $X \in \mathbb{B}$. The following statements are equivalent:*

⁷Recall the definition of weak equality in Def. 3.3.

⁸Note that in the left coproduct of (4.1) an object \mathcal{D}_i occurs as often as there are edges d leaving i in \mathbb{I} . Moreover, $\subseteq_{i,d}$ in (4.2) denotes the embedding of \mathcal{D}_i into its appropriate copy, namely the source of \mathcal{D}_d .

- $\forall i, j \in \mathbb{I}_0 : \forall z \in \mathcal{D}_i(X), \forall z' \in \mathcal{D}_j(X)$: There are no two disjoint proper mapping paths in \mathcal{D} connecting z and z' .
- $\forall z, z' \in \overline{\mathcal{D}}_2(X) = \coprod_{j \in \mathbb{I}_0} \mathcal{D}_j(X)$: There are no two disjoint proper mapping paths in $\overline{\mathcal{D}}$ connecting z and z' . \square

The main part of the proof of Theorem 3.5 is to carry over the VK property for the coequalizer of (4.1) to its underlying colimiting diagram for \mathcal{D} .

Lemma 4.7. *For $\mathbb{G} := \text{Set}^{\mathbb{B}}$, the cocone (3.1) is VK if and only if the cocone*

$$\coprod_{d \in \mathbb{I}_1} \mathcal{D}_s(d) \begin{array}{c} \xrightarrow{\bar{id}} \\ \xrightarrow{\bar{D}_d} \\ \xrightarrow{\bar{\kappa}'} \end{array} \coprod_{j \in \mathbb{I}_0} \mathcal{D}_j \xrightarrow{\bar{\kappa}} S \quad (4.3)$$

resulting from constructing the coequalizer in (4.1) is VK.

Proof: Let $\kappa^* : \mathbb{G} \downarrow S \rightarrow \mathbb{G}^{\mathbb{I}} \downarrow \mathcal{D}$ be the functor introduced in (3.3) and $\bar{\kappa}^* : \mathbb{G} \downarrow S \rightarrow \mathbb{G}^2 \downarrow \overline{\mathcal{D}}$ be the corresponding functor for the colimiting cocone in (4.3). Using (2.1) and (2.2), one can show that for each cartesian $\tau : \mathcal{E} \Rightarrow \mathcal{D}$ the squares

$$\begin{array}{ccc} \coprod_{d \in \mathbb{I}_1} \mathcal{E}_s(d) & \xrightarrow{\bar{id}} & \coprod_{i \in \mathbb{I}_0} \mathcal{E}_i & & \coprod_{d \in \mathbb{I}_1} \mathcal{E}_s(d) & \xrightarrow{\bar{\mathcal{E}}_d} & \coprod_{j \in \mathbb{I}_0} \mathcal{E}_j \\ \downarrow \coprod_{d \in \mathbb{I}_1} \tau_s(d) & & \downarrow \coprod_{i \in \mathbb{I}_0} \tau_i & & \downarrow \coprod_{d \in \mathbb{I}_1} \tau_s(d) & & \downarrow \coprod_{j \in \mathbb{I}_0} \tau_j \\ \coprod_{d \in \mathbb{I}_1} \mathcal{D}_s(d) & \xrightarrow{\bar{id}} & \coprod_{i \in \mathbb{I}_0} \mathcal{D}_i & & \coprod_{d \in \mathbb{I}_1} \mathcal{D}_s(d) & \xrightarrow{\bar{D}_d} & \coprod_{j \in \mathbb{I}_0} \mathcal{D}_j \end{array}$$

are pullbacks, i.e., there is the assignment $\tau \mapsto (\coprod_{d \in \mathbb{I}_1} \tau_s(d), \coprod_{i \in \mathbb{I}_0} \tau_i)$. It can be shown with elementary arguments that it extends to an equivalence of categories:

$$\phi : \mathbb{G}^{\mathbb{I}} \downarrow \mathcal{D} \cong \mathbb{G}^2 \downarrow \overline{\mathcal{D}}.$$

Moreover, the colimit construction principle, see (4.1), yields commutativity of

$$\begin{array}{ccc} & \mathbb{G} \downarrow S & \\ \kappa^* \swarrow & & \searrow \bar{\kappa}^* \\ \mathbb{G}^{\mathbb{I}} \downarrow \mathcal{D} & \xrightarrow{\phi} & \mathbb{G}^2 \downarrow \overline{\mathcal{D}} \end{array}$$

up to natural isomorphism, hence, by Def. 3.2, the desired result. \square

4.4. Combining the Results. We are now ready to prove Theorem 3.5 for disjoint proper mapping paths. This follows by combining Lemma 4.7, Corollary 4.5 and Lemma 4.6. Afterwards we can get rid of disjointness by showing that any two proper mapping paths connecting the same two elements also admit two disjoint proper paths (probably connecting two different elements). \square

5. COUNTEREXAMPLES IN OTHER CATEGORIES

The valuable implication in the equivalence statement of Theorem 3.5 is "Path-Uniqueness implies the Van Kampen Property". In this section, we show that this implication immediately breaks, if we leave the universe of presheaf topoi, i.e. there are simple counter-examples in categories that are similar to, but don't meet all axioms of presheaf topoi. We separate the examples depending on whether unit η or counit ε of adjunction $\kappa_* \dashv \kappa^*$ fails to be isomorphic (cf. Def.3.2 and ensuing remarks) although path uniqueness holds.

For the sake of completeness we finally give an example for a violation of the reverse direction of the equivalence in Theorem 3.5. It illustrates a situation of a Van Kampen cocone, although path uniqueness does not hold.

5.1. Path Uniqueness, but no Isomorphic Unit. Consider the category $HSet$, whose objects are sets, where in each set at most one element may be highlighted (we also say "marked"). We denote such a set with (x, A) (indicating that $x \in A$ is marked). If no element is highlighted, we write (\perp, A) . Formally these sets are *partial* algebras [4, 30, 31, 32] w.r.t. the algebraic signature Σ with one sort and one constant symbol h ("highlighted"), such that in any Σ -algebra with carrier set A there is a constant, which can be understood as a partial map $h : \{*\} \rightarrow A$, i.e. there may or may not be a marked element depending on whether h is defined for $*$ or not. A morphism between (x, A) and (y, B) is a total mapping $f : A \rightarrow B$ for which the additional property

$$x \neq \perp \Rightarrow (y \neq \perp \text{ and } f(x) = y)$$

holds. It can be shown [4] that $HSet$ possesses all limits and colimits.

In Fig.2, the bottom face is a pushout (note that the one element set $\{*\}$ arises from the fact that there can not be more than one marked element). The behaviour of each mapping should be obvious. Clearly, the back faces are pullbacks. Although the top face is a pushout, the front faces fail to be pullbacks. Path uniqueness for (h_1, h_2) is satisfied trivially.

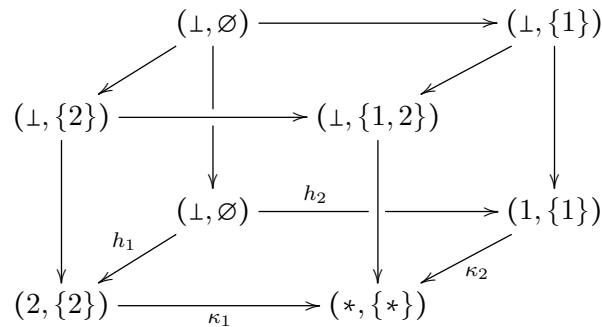


Figure 2: Path Uniqueness, but not VK, I

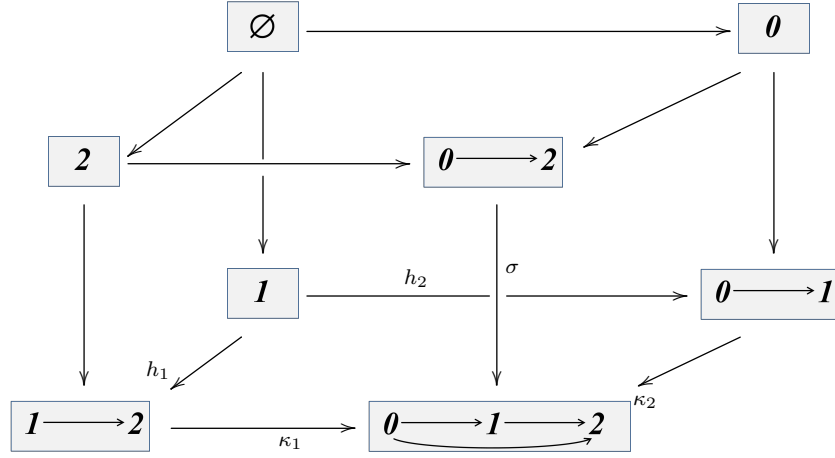


Figure 3: Path Uniqueness, but not VK, II

5.2. Path Uniqueness but no Isomorphic Co-Unit. By the remarks in Sect.3, this can only be the case, if the pullback functor possesses no right-adjoint. For this, we consider the category $\mathbb{C} = CAT$ of all small categories together with functors between them. The following example in \mathbb{C} simultaneously shows (1) a non-VK-pushout with path uniqueness and (2) that the pullback functor has no right-adjoint. In Fig.3 each shaded rectangle shows a category, e.g. in the bottom there is a category with exactly one element 1, two categories with one non-identity arrow between objects 0 and 1, 1 and 2, resp., and a category with three elements 0,1, and 2 where the arrow from 0 to 2 is the composition of the other two. In all cases, identity arrows are not shown. Arrows between the shaded rectangles are functors that map according to the numbering. This example has already been discussed in [24].

Obviously, the bottom square is a pushout in \mathbb{C} and the back faces are pullbacks (\emptyset denotes the empty category). Moreover, the front faces are pullbacks, but the top face fails to be a pushout (the pushout should be the category consisting of two objects 0 and 2 and no non-identity arrow), hence ε is not an isomorphism. The pullback functor σ^* maps the bottom square to the top square. If it would possess a right-adjoint, it would preserve the bottom colimit, which is not the case.

Clearly, path uniqueness holds in span (h_1, h_2) , because both functors are injective on objects and arrows, resp. Note that we treat \mathbb{C} as a many-sorted algebra with carrier sets $O(\text{bjects})$ and $(Hom(x, y))_{x, y \in O}$ (Hom-Sets) and (in contrast to graphs) with a family of binary operations $(\circ_{x, y, z} : Hom(y, z) \times Hom(x, y) \rightarrow Hom(x, z))_{x, y, z \in O}$, a family of constants $(id_x \in Hom(x, x))_{x \in O}$, and the appropriate monoidal axioms for neutrality (of identities) and associativity (of operations $\circ_{-, -, -}$).

5.3. Van Kampen holds, but Path Uniqueness is Violated. This requires an artificial and radical narrowing of the size of the underlying category. We consider a category \mathbb{C} which has sets X with two constants 0 and 1 (actually an algebra for a signature with one sort and two constant symbols) subject to the following axiom

$$\forall x \in X : x = 0 \vee x = 1.$$

Let sets X and Y with constants $0^X, 1^X$ and $0^Y, 1^Y$ be given, then an arrow from X to Y is a mapping $f : X \rightarrow Y$, which preserves constants, i.e. $f(0^X) = 0^Y$ and $f(1^X) = 1^Y$. It is not forbidden that the two constants coincide, such that the above equation reduces this category to two sets $2 := \{0, 1\}$ and $1 := \{01\}$ (in set 1, the constants coincide). It can further be observed that the only non-identity arrow is $2 \rightarrow 1$. Furthermore, the only pullback of a co-span with two non-identity arrows is

$$\begin{array}{ccc} 2 & \xlongequal{\quad} & 2 \\ \parallel & & \downarrow \\ 2 & \longrightarrow & 1 \end{array}$$

It can even be shown that this (very small) category has all limits and colimits (e.g. the initial object is 2, terminal object is 1).

We can now pick up the coequalizer example (3.4) of Sect.3.1. There is the coequalizer diagram $1 \rightrightarrows 1 \rightarrow (1 =: S)$ where all arrows are identities. As pointed out in Example 3.4, 1 path uniqueness is violated.

In each pullback pair

$$\begin{array}{ccc} A & \rightrightarrows & B \\ \downarrow & & \downarrow \\ 1 & \rightrightarrows & 1 \end{array}$$

we must have $A = B$ and the two top mappings must both be identities. Note that the non-identical bijection of $\{0, 1\}$ as in (3.4) can no longer occur. Thus, only two such pullback pairs exist. There are also exactly two objects in $\mathbb{C} \downarrow S$, namely id_1 and $2 \rightarrow 1$, and it is easy to see that they correspond to each other via κ^* and κ_* . Hence the coequalizer has the Van Kampen property, although path uniqueness is violated.

6. PRACTICAL GUIDELINES

A main use case of the previous results can be found in software engineering and especially in model-driven software designs: Components of diagrams are models (e.g. data models or metamodels which govern the admissible structure of models), morphisms are relations between the models. As described in the introduction, model assembly is often important (cf. Sect.3.3). It shall not only be carried out on a syntactical level (models), but in the same way on the semantical level (instances) such that assembled instances can correctly be decomposed into their original instances by pullback.

It is a goal to efficiently verify whether *compositionality* holds, i.e. whether the Van Kampen property is satisfied. Since model composition always requires computation of colimits, VK-verification at the same time is desirable. In this section we will describe (1) an efficient colimit computation and (2) how to simultaneously check the VK-property.

To further reduce verification effort, we will first look for criteria to decide, for a given diagram, if VK holds or not, without checking explicitly the path conditions of Theorem 3.5. Moreover, we will show that – in cases where the path condition is needed – one does not need to check the condition for *all* $i, j \in \mathbb{I}_0$ but only for a smaller subset of indices. All investigations lead to a decision diagram, which guides a modeler who has to decide whether a given diagram has the VK property or not. This diagram is given in the end of this section, see Fig. 4.

All additional constructions are elementary and will not be elaborated in detail. Instead we refer the reader to the corresponding technical report [12].

6.1. Relevant Types of Mapping Paths. We will discuss now what kinds of paths and what pairs of paths we really need to check in practice. The attentive reader may have noticed already that properness excludes cycles w.r.t. path segments but does not exclude cycles w.r.t. elements. Let $P = [(y_0, \delta_0, y_1), (y_1, \delta_1, y_2), \dots, (y_{n-1}, \delta_{n-1}, y_n)]$ be a mapping path in a diagram $\mathcal{D} : \mathbb{I} \rightarrow \text{Set}^{\mathbb{B}}$ with finite \mathbb{I} . By ι_i , we denote the unique vertex in \mathbb{I}_0 with $y_i \in \mathcal{D}_{\iota_i}$.

Definition 6.1. A mapping path is called *inner-cycle free*, if for all indices $0 \leq i < j \leq n$ with $j - i \leq n - 1$: $y_i \neq y_j$.

Empty mapping paths or paths of length 1 are inner-cycle free by definition. Note, that we allow P to be an "outer" cycle, i.e., $y_0 = y_n$. An elementary construction [12] shows that each path can be reduced to an inner-cycle free non-empty (sub-)path. The reduction preserves properness and disjointness of mapping paths. Moreover, one can easily see that two different proper paths from z to z' even yield two *disjoint* proper paths (probably between other elements). Thus we obtain the following corollary of Theorem 3.5:

Corollary 6.2. Let $\mathbb{G} = \text{Set}^{\mathbb{B}}$ be a presheaf topos and $\mathcal{D} : \mathbb{I} \rightarrow \mathbb{G}$ be a diagram with \mathbb{I} a finite directed multigraph. Let

$$\mathcal{D} \xrightarrow{\kappa} \Delta S$$

be a colimiting cocone. The following are equivalent:

- (1) The cocone is a Van Kampen cocone
- (2) $\forall X \in \mathbb{B}, i, j \in \mathbb{I}_0, z \in \mathcal{D}_i(X), z' \in \mathcal{D}_j(X)$: There are no two different proper paths from z to z'
- (3) $\forall X \in \mathbb{B}, i, j \in \mathbb{I}_0, z \in \mathcal{D}_i(X), z' \in \mathcal{D}_j(X)$: There are no two disjoint proper paths from z to z'
- (4) $\forall X \in \mathbb{B}, i, j \in \mathbb{I}_0, z \in \mathcal{D}_i(X), z' \in \mathcal{D}_j(X)$: There are no two disjoint inner-cycle free proper paths from z to z'

In addition to a better variety of VK characterisations, this result also provides a high degree of freedom for the implementation of algorithms: The result does not depend on whether an algorithm generates only disjoint pairs of paths, or also builds paths with inner cycles. However, every algorithm based on Corollary 6.2 still has to traverse all components $(\mathcal{D}_i)_{i \in \mathbb{I}_0}$. The next sections will explain why the traversal of components can significantly be reduced.

6.2. Cyclic Shape Graphs. A *directed cycle* in \mathbb{I} is a set of pairwise distinct edges d_0, \dots, d_{n-1} in \mathbb{I}_1 for some $n \geq 1$ with $t(d_{i-1}) = s(d_{i \bmod n})$ ($i \in \{1, \dots, n\}$). If $n = 1$, the cycle is also called a loop (cf. Example 3.4, 3). Let $\text{ends}(d) = \{s(d), t(d)\}$ be the set of endpoints of an edge d , then an *undirected cycle* in \mathbb{I} is a set of pairwise distinct edges d_0, \dots, d_{n-1} in \mathbb{I}_1 for some $n \geq 2$ with $\text{ends}(d_{i-1}) \cap \text{ends}(d_{i \bmod n}) \neq \emptyset$ ($i \in \{1, \dots, n\}$). An example for an undirected cycle is the shape graph $\mathfrak{2}$ and also the shape graph in the example of Sect.3.3.

An important observation is that VK is violated, if \mathbb{I} possesses a directed cycle

$$p = (i_0 \xrightarrow{d_0} i_1 \xrightarrow{d_1} \dots \xrightarrow{d_{n-1}} i_n = i_0)$$

and if for some $X \in \mathbb{B}$ and some $i \in \{i_0, \dots, i_n\}$ the component $\mathcal{D}_i(X)$ is a finite set. To see this, let w.l.o.g. $i = i_0$ and

$$\mathcal{D}_p := \mathcal{D}_{d_{n-1}} \circ \dots \circ \mathcal{D}_{d_1} \circ \mathcal{D}_{d_0} : \mathcal{D}_{i_0}(X) \rightarrow \mathcal{D}_{i_n}(X) = \mathcal{D}_{i_0}(X).$$

be the corresponding composed function. Obviously, there exist $y \in \mathcal{D}_{i_0}(X)$ and $1 \leq k \leq |\mathcal{D}_{i_0}(X)|$ such that $\mathcal{D}_p^k(y) = y$, where the mappings can be chosen such that this results in a non-empty proper mapping path connecting y with itself: Because the empty path also connects y with itself (cf. Def.3.3), VK is violated by Cor.6.2.

6.3. Specialized Construction of Colimits. In this section, we prepare a general and more efficient algorithm for VK verification, which runs in the background of a colimit computation. For this we need to distinguish between different characteristics of shape graph \mathbb{I} and derive from that a specialized colimit construction.

The universal construction recipe for colimits (4.1) shows how to construct the colimit of any diagram in a uniform way by means of a coequalizer. It allows to prove general results about colimits (as we have demonstrated in the previous sections). Especially, in case of graphs \mathbb{I} with infinite descending chains and/or with directed cycles, this universal recipe is the best we have. E.g. colimit construction of

$$\mathcal{D}_1 \begin{array}{c} \xrightarrow{\mathcal{D}_d} \\ \xleftarrow{\mathcal{D}_{d'}} \end{array} \mathcal{D}_2,$$

and its VK verification is based on mapping paths within the coproduct $\mathcal{D}_1 + \mathcal{D}_2$ and there is no way of minimizing the space for investigation.

A first step in simplifying colimit computation and VK verification can be found, if we consider coequalizers. They can be investigated within a smaller space: A coequalizer is not VK, if we can find for the corresponding diagram $\mathcal{D} : \mathbf{2} \rightarrow \mathit{Set}^{\mathbb{B}}$ a sort $X \in \mathbb{B}$ and an element $y \in \mathcal{D}_1(X)$ such that $\mathcal{D}_d(y) = \mathcal{D}_{d'}(y)$, thus reducing investigations to \mathcal{D}_1 . However, even if \mathcal{D}_d and $\mathcal{D}_{d'}$ do have sortwise disjoint images, VK may be violated. Note, that those image disjoint diagrams are exactly the diagrams we obtain when constructing pushouts, in the traditional way, by means of sums and coequalizer. To have VK we can require, in addition, that \mathcal{D}_d and $\mathcal{D}_{d'}$ are monic, since then $[\mathcal{D}_d, \mathcal{D}_{d'}]$ is monic and hence the pushout along this mono is VK (because each topos is adhesive [15, 24]). In these special cases, this provides again significant simplification.

Even in the cases left unclear, the check of the VK property for coequalizers must not utilize the condition in Cor.6.2, which is based on the universal construction recipe (4.1) for colimits⁹. Instead, we can use directly the condition in Corollary 4.5, i.e. we need to check the condition in Cor.6.2 only for the case $i = j = 2$, thus reducing investigations to \mathcal{D}_2 . We will see in the forthcoming parts that all these effects can often be used in more general cases to reduce analysis effort.

Beside these effects, there may be components that do not contribute to the construction of the colimit at all. In many cases, this simplifies colimit construction, because it is not necessary to compute a quotient of the *entire* coproduct $\coprod_{j \in \mathbb{I}_0} \mathcal{D}_j$. Moreover, we will investigate how certain further assumptions on the properties of arrows in \mathcal{D} (image-disjointness, injectivity) also simplify the algorithm. We will discuss some further examples to motivate the announced specialized and practical construction of colimits.

⁹Note, that we could apply the universal construction recipe again to the diagram in (4.1) and so on.

Since the case of directed cycles can immediately be handled in practical situations¹⁰, we assume from now on that \mathbb{I} is finite and has no directed cycles.

Irrelevant components: For all indices in \mathbb{I} with no incoming and exactly one outgoing edge, the corresponding component does not contribute to the construction of the colimit. Typical examples are

$$\mathcal{D}_1 \xrightarrow{\mathcal{D}_d} \mathcal{D}_2 \qquad \mathcal{D}_1 \xrightarrow{\mathcal{D}_{d_1}} \mathcal{D}_3 \xleftarrow{\mathcal{D}_{d_2}} \mathcal{D}_2,$$

(in an arbitrary category). For the left diagram \mathcal{D}_2 can be taken as the colimit object and we can set $\kappa_2 := id_{\mathcal{D}_2}$, $\kappa_1 := \mathcal{D}_d$. For the right diagram \mathcal{D}_3 can serve as colimit object and we have $\kappa_3 := id_{\mathcal{D}_3}$, $\kappa_1 := \mathcal{D}_{d_1}$, $\kappa_2 := \mathcal{D}_{d_2}$.

Jump-over components: For all indices in \mathbb{I} with exactly one incoming and one outgoing edge we can jump over the corresponding component. As examples, we consider the diagrams

$$\mathcal{D}_0 \xrightarrow{\mathcal{D}_{d_1}} \mathcal{D}_1 \xrightarrow{\mathcal{D}_{d_2}} \mathcal{D}_2 \qquad \mathcal{D}_3 \xleftarrow{\mathcal{D}_{d_3}} \mathcal{D}_0 \xrightarrow{\mathcal{D}_{d_1}} \mathcal{D}_1 \xrightarrow{\mathcal{D}_{d_2}} \mathcal{D}_2,$$

(in an arbitrary category). For the left diagram, \mathcal{D}_2 can be taken as the colimit object and we have $\kappa_2 := id_{\mathcal{D}_2}$, $\kappa_1 := \mathcal{D}_{d_2}$, $\kappa_0 := \mathcal{D}_{d_2} \circ \mathcal{D}_{d_1}$. For the right diagram the colimit object is obtained by the pushout of $\mathcal{D}_{d_2} \circ \mathcal{D}_{d_1} : \mathcal{D}_0 \rightarrow \mathcal{D}_2$ and $\mathcal{D}_{d_3} : \mathcal{D}_0 \rightarrow \mathcal{D}_3$. The missing injections are given by $\kappa_1 := \kappa_2 \circ \mathcal{D}_{d_1}$ and $\kappa_0 := \kappa_2 \circ \mathcal{D}_{d_2} \circ \mathcal{D}_{d_1} (= \kappa_3 \circ \mathcal{D}_{d_3})$.

Minimal components: We consider diagrams for coequalizer and pushouts, respectively,

$$\mathcal{D}_1 \begin{array}{c} \xrightarrow{\mathcal{D}_d} \\ \xrightarrow{\mathcal{D}_{d'}} \end{array} \mathcal{D}_2 \qquad \mathcal{D}_1 \xleftarrow{\mathcal{D}_d} \mathcal{D}_0 \xrightarrow{\mathcal{D}_{d'}} \mathcal{D}_2$$

and the corresponding diagrams according to the universal recipe (4.1)

$$\mathcal{D}_1 + \mathcal{D}_1 \begin{array}{c} \xrightarrow{[\varepsilon_1, \varepsilon_1]} \\ \xrightarrow{[\varepsilon_2 \circ \mathcal{D}_d, \varepsilon_2 \circ \mathcal{D}_{d'}]} \end{array} \mathcal{D}_1 + \mathcal{D}_2 \qquad \mathcal{D}_0 + \mathcal{D}_0 \begin{array}{c} \xrightarrow{[\varepsilon_0, \varepsilon_0]} \\ \xrightarrow{[\varepsilon_1 \circ \mathcal{D}_d, \varepsilon_2 \circ \mathcal{D}_{d'}]} \end{array} \mathcal{D}_0 + \mathcal{D}_1 + \mathcal{D}_2$$

In case of coequalizer we construct, usually, a quotient of \mathcal{D}_2 and not of $\mathcal{D}_1 + \mathcal{D}_2$ and, in case of pushouts we construct a quotient of $\mathcal{D}_1 + \mathcal{D}_2$ and not of $\mathcal{D}_0 + \mathcal{D}_1 + \mathcal{D}_2$. Also in the example in Figure 1 we factorize the sum $\mathcal{D}_1 + \mathcal{D}_2 + \mathcal{D}_3$ and not the sum of all 6 components. In all three cases we build first the coproduct of all minimal components (see Def. 6.3) and construct then a quotient of this restricted coproduct.

Definition 6.3 (Minimal Components). For a finite directed multigraph \mathbb{I} we denote by $Min(\mathbb{I})$ the set of all (local) minimal indices, i.e., of all vertices in \mathbb{I}_0 without outgoing edges. For a diagram $\mathcal{D} : \mathbb{I} \rightarrow Set^{\mathbb{B}}$ we say that \mathcal{D}_i is a minimal component if $i \in Min(\mathbb{I})$.

Since \mathbb{I} is finite and has no directed cycles, each index is either minimal or there exists a non-empty finite sequence of edges to at least one minimal index. This fact will be used several times in the sequel. To construct the colimit of a diagram with finite \mathbb{I} with no directed cycles we need, in practice, only the minimal components as outlined below.

¹⁰ ... where, presumably, components are finite artefacts ...

Branching components: The essence in constructing the colimit of a diagram is to converge diverging branches in the diagram (in a minimal way). In presheaf topoi this can be done by sortwise identifying certain elements. What elements, however, have to be identified?

In case of coequalizers we have to identify for all sorts $X \in \mathbb{B}$ and all $y \in \mathcal{D}_1(X)$ the two elements $\mathcal{D}_d(y)$ and $\mathcal{D}_{d'}(y)$ in $\mathcal{D}_2(X)$. These primary identifications induce further identifications when we construct the smallest congruence in \mathcal{D}_2 comprising all these primary identifications. For pushouts we have to identify for all $X \in \mathbb{B}$ and all $y \in \mathcal{D}_0(X)$ the two elements $\mathcal{D}_d(y)$ and $\mathcal{D}_{d'}(y)$ seen as elements in $\mathcal{D}_1(X) + \mathcal{D}_2(X)$. In this case, we construct then the smallest congruence in $\mathcal{D}_1 + \mathcal{D}_2$ comprising all the primary identifications.

Now we look at slightly more general diagrams. First, we consider three parallel arrows.

$$\begin{array}{ccc} & \mathcal{D}_{d_1} & \\ & \curvearrowright & \\ \mathcal{D}_1 & \xrightarrow{\mathcal{D}_{d_2}} & \mathcal{D}_2 \\ & \curvearrowleft & \\ & \mathcal{D}_{d_3} & \end{array}$$

In this case, we have to identify for all sorts $X \in \mathbb{B}$ and all $y \in \mathcal{D}_1(X)$ the three elements $\mathcal{D}_{d_1}(y)$, $\mathcal{D}_{d_2}(y)$ and $\mathcal{D}_{d_3}(y)$ in $\mathcal{D}_2(X)$, and then we construct the smallest congruence in \mathcal{D}_2 comprising these identifications. Second, we consider two generalizations of pushouts

$$\begin{array}{ccc} & \mathcal{D}_I & \\ \mathcal{D}_{is} \swarrow & & \searrow \mathcal{D}_{ib} \\ \mathcal{D}_S & \xleftarrow{\mathcal{D}_m} \mathcal{D}_P \xrightarrow{\mathcal{D}_r} & \mathcal{D}_B \end{array} \qquad \begin{array}{ccccc} & \mathcal{D}_1 & \xleftarrow{\mathcal{D}_{d_{-13}}} & \mathcal{D}_{13} & \xrightarrow{\mathcal{D}_{d_{13}}} & \mathcal{D}_3 \\ \mathcal{D}_{d_{-12}} \uparrow & & \mathcal{D}_{d_{12}} & \xrightarrow{\quad} & \mathcal{D}_2 & \xleftarrow{\mathcal{D}_{d_{-23}}} & \mathcal{D}_{23} \\ & & & & & & \uparrow \mathcal{D}_{d_{23}} \end{array}$$

A situation, as in the left diagram, appears, for example, if we want to avoid that the instantiation of a "parameterized specification" $\mathcal{D}_P \xrightarrow{\mathcal{D}_r} \mathcal{D}_B$ via a "match" $\mathcal{D}_P \xrightarrow{\mathcal{D}_m} \mathcal{D}_S$ generates two copies of a specification \mathcal{D}_I that had been imported as well by the "body" \mathcal{D}_B of the parameterized specification as by the "actual parameter" \mathcal{D}_S . The diagram on the right is taken from the example in Figure 1.

In the left diagram we have to identify for all $X \in \mathbb{B}$ and all $y \in \mathcal{D}_P(X)$ the two elements $\mathcal{D}_m(y)$ and $\mathcal{D}_r(y)$ seen as elements in $\mathcal{D}_S(X) + \mathcal{D}_B(X)$. In addition, we have to identify for all $z \in \mathcal{D}_I(X)$ the two elements $\mathcal{D}_{is}(z)$ and $\mathcal{D}_{ib}(z)$, again seen as elements in $\mathcal{D}_S(X) + \mathcal{D}_B(X)$.

In the right diagram, we have, analogously, that the elements in \mathcal{D}_{12} force identifications of elements in \mathcal{D}_1 and \mathcal{D}_2 , seen as elements in $\mathcal{D}_1 + \mathcal{D}_2 + \mathcal{D}_3$, the elements \mathcal{D}_{13} force identifications of elements in \mathcal{D}_1 and \mathcal{D}_3 , seen as elements in $\mathcal{D}_1 + \mathcal{D}_2 + \mathcal{D}_3$, and the elements in \mathcal{D}_{23} force identifications of elements in \mathcal{D}_2 and \mathcal{D}_3 , seen as elements in $\mathcal{D}_1 + \mathcal{D}_2 + \mathcal{D}_3$.

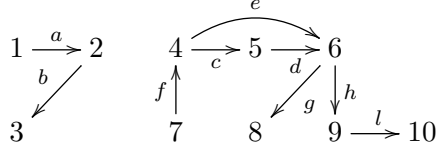
Generalizing the examples we want to coin the following definition.

Definition 6.4 (Branching Components). For a finite directed multigraph \mathbb{I} we denote by $Br(\mathbb{I})$ the set of all *branching indices*, i.e., of all indices with, at least, two outgoing edges. For a diagram $\mathcal{D} : \mathbb{I} \rightarrow Set^{\mathbb{B}}$ we say that \mathcal{D}_i is a *branching component* if $i \in Br(\mathbb{I})$.

A sequence of edges $p = (i_0 \xrightarrow{d_0} i_1 \xrightarrow{d_1} \dots \xrightarrow{d_{n-1}} i_n)$ in \mathbb{I} is called a *branch*, if $i_0 \in Br(\mathbb{I})$ and $i_n \in Min(\mathbb{I})$.

Note, that $Br(\mathbb{I})$ and $Min(\mathbb{I})$ are disjoint by definition. Thus any branch has at least length 1. Note further, that branching indices can be connected in \mathbb{I} , in contrast to minimal indices. To illustrate our discussion and definitions we consider a simple toy example.

Example 6.5. Let $\mathbb{I} =$



Here we have $Min(\mathbb{I}) = \{3, 8, 10\}$, $Br(\mathbb{I}) = \{4, 6\}$ and the four branches $(4 \xrightarrow{c} 5 \xrightarrow{d} 6 \xrightarrow{g} 8)$, $(4 \xrightarrow{c} 5 \xrightarrow{d} 6 \xrightarrow{h} 9 \xrightarrow{l} 10)$, $(4 \xrightarrow{e} 6 \xrightarrow{g} 8)$, and $(4 \xrightarrow{e} 6 \xrightarrow{h} 9 \xrightarrow{l} 10)$.

The indices 1 and 7 are irrelevant and we can jump over the indices 5 and 9. We can also jump over the index 2, but since 1 is irrelevant also 2 becomes irrelevant. Be aware that the edges c, d, e constitute an undirected cycle in \mathbb{I} .

Branching indices give rise to special positions in mapping paths:

Definition 6.6 (Branching Position in Proper Paths). For a pair of subsequent segments

$$(y_{j-1}, d^{op}, y_j), (y_j, d', y_{j+1})$$

of a proper mapping path P (hence $d \neq d'$) the position j is called a *branching position* of P . Consequently ι_j is a branching index of \mathbb{I} .

A specialized construction of colimits: Now, we have everything at hand to describe a construction of colimits in presheaf topoi. Let \mathbb{I} be a finite directed multigraph with no directed cycles. Then we can construct the colimit of a diagram $\mathcal{D} : \mathbb{I} \rightarrow Set^{\mathbb{B}}$ as follows:

- (1) Construct the coproduct $\coprod_{i \in Min(\mathbb{I})} \mathcal{D}_i$ (by sortwise coproducts in Set).
- (2) For each pair $p = (i \xrightarrow{d} \dots \rightarrow j)$, $p' = (i \xrightarrow{d'} \dots \rightarrow j')$ of branches in \mathbb{I} with common source $i \in Br(\mathbb{I})$ and $d \neq d'$, and for each sort $X \in \mathbb{B}$ there is the set

$$\approx_X^{p,p'} := \{(\subseteq_j(\mathcal{D}_p(y)), \subseteq_{j'}(\mathcal{D}_{p'}(y))) \mid y \in \mathcal{D}_i(X)\}$$

of pairs (primary identifications) in $\coprod_{i \in Min(\mathbb{I})} \mathcal{D}_i$.¹¹ Each pair is represented by a mapping path (called a *primary mapping path*) connecting the pair's components¹². By \approx_X we denote the union of all those sets $\approx_X^{p,p'}$ for sort X . This results in a family $\approx = (\approx_X)_{X \in \mathbb{B}}$ of binary relations in $\coprod_{i \in Min(\mathbb{I})} \mathcal{D}_i$.

- (3) Construct the smallest congruence $\cong = (\cong_X)_{X \in \mathbb{B}}$ in $\coprod_{i \in Min(\mathbb{I})} \mathcal{D}_i$ which comprises \approx by enlargement with transitive (i.e. concatenation of primary mapping paths) and reflexive (empty mapping paths) closure¹³.
- (4) Construct the colimit object as the sortwise quotient $(\coprod_{i \in Min(\mathbb{I})} \mathcal{D}_i) / \cong$ and get, in such a way, also the canonical morphisms $[]_{\cong} : \coprod_{i \in Min(\mathbb{I})} \mathcal{D}_i \rightarrow (\coprod_{i \in Min(\mathbb{I})} \mathcal{D}_i) / \cong$.
- (5) The colimiting cocone of diagram \mathcal{D} is given by

$$\mathcal{D} \xrightarrow{\kappa} \left(\coprod_{i \in Min(\mathbb{I})} \mathcal{D}_i \right) / \cong \tag{6.1}$$

¹¹ Recall the previous definition of \mathcal{D}_p in Sect. 6.2 as the compositions of all \mathcal{D}_d with d an arrow of path p

¹²It can be shown inductively over the path length that each pair can even be represented by a path *with exactly one branching position*.

¹³ \approx is already symmetric by definition and the transitive closure preserves symmetry. It is then easy to see that compatibility with operation symbols holds.

where $\kappa_i := []_{\cong} \circ \subseteq_i$ for all minimal indices $i \in \text{Min}(\mathbb{I})$ and $\kappa_i := \kappa_j \circ \mathcal{D}_p$ for all other indices $i \in \mathbb{I}_0 \setminus \text{Min}(\mathbb{I})$, where $p = (i \rightarrow \dots \rightarrow j)$ is an edge sequence from i to $j \in \text{Min}(\mathbb{I})$. A detailed proof for the validity of (6.1) is given in [12]. Note, that the definition of κ_i is independent of the choice of p since we have, by construction, $\kappa_j \circ \mathcal{D}_p = \kappa_{j'} \circ \mathcal{D}_{p'}$ for all branches $p = (i \rightarrow \dots \rightarrow j)$, $p' = (i \rightarrow \dots \rightarrow j')$ in \mathbb{I} with a common source.

The main advantage of this construction is that mapping paths are now computed traversing branching components only. These components, however, are often just tiny "connectors" as in Fig.1. Note, that all the components \mathcal{D}_i with $i \in \text{Min}(\mathbb{I})$ not being the target of any branch are not affected by the quotient construction, i.e. congruence classes $[z]_{\cong}$ are singletons for all $z \in \mathcal{D}_i(X)$. In Example 6.5 this is the case for index 3. Let $Af(\mathbb{I})$ denote the set of all affected minimal indices. We could then be even more specific and construct the colimit object as

$$\left(\coprod_{i \in \text{Min}(\mathbb{I}) \setminus Af(\mathbb{I})} \mathcal{D}_i \right) + \left(\coprod_{i \in Af(\mathbb{I})} \mathcal{D}_i \right) / \cong \quad (6.2)$$

6.4. Efficient Checking of the Van Kampen Property. Based on (6.2) we can conclude, independent of Corollary 6.2, that a diagram $\mathcal{D} : \mathbb{I} \rightarrow \text{Set}^{\mathbb{B}}$ is VK if \mathbb{I} , in addition of being finite and having no directed cycles, does not have branching indices either. In this case, $Af(\mathbb{I})$ is empty and the colimit of the diagram is simply given by the coproduct of all minimal components, thus the VK property of the diagram is ensured by the VK property of coproducts (extensivity) and pullback composition. In the presence of branching, however, we do not have VK for free. We have to check one of the conditions in Cor.6.2.

The specialized colimit construction (6.2) suggests another practical relevant possibility to reduce our effort for checking VK in case \mathbb{I} has no directed cycles. In applications that deal with nets of software components (e.g. multimodels), there is usually only one type of relation between the components: Relations either specify sameness of model elements, versions of one model element in evolving environments, or elements to be preserved when applying transformation rules [3]. Thus, rarely will it be the case that there are two or more morphisms in the same direction between two given components. An even weaker and also reasonable claim for two different relations is that they don't interfere in common codomains, thus the following definition is not too restrictive:

Definition 6.7 (Image-Disjointness). A diagram $\mathcal{D} : \mathbb{I} \rightarrow \text{Set}^{\mathbb{B}}$ is called *image disjoint*, if for each pair of different branches $p = (i \rightarrow \dots \rightarrow j)$, $p' = (i \rightarrow \dots \rightarrow j')$ in \mathbb{I} starting in the same branching index i and all elements $y \in \mathcal{D}_i(X)$, $X \in \mathbb{B}$ we have $\mathcal{D}_p(y) \neq \mathcal{D}_{p'}(y)$.

Fact 6.8. If \mathcal{D} is not image-disjoint, the colimit $\mathcal{D} \Rightarrow \Delta S$ does not have the Van Kampen property.

Proof. The two different branches p, p' and $y \in \mathcal{D}_i(X)$ with $\mathcal{D}_p(y) = \mathcal{D}_{p'}(y)$ yield two different mapping paths from y to $\mathcal{D}_p(y)$. Thus the result follows from Cor.6.2. \square

If there are no undirected cycles in \mathbb{I} , then we have image disjointness for free, because we always assume that all components \mathcal{D}_i are pairwise disjoint. If there are undirected cycles in \mathbb{I} , as in the case of coequalizers, for example, it can happen that $j = j'$. Thus we have to test, first, for image disjointness before the "different paths criterion for paths connecting affected minimal components" below can be applied. Note, that image disjointness implies that we have $\mathcal{D}_p(y) \neq \mathcal{D}_{p'}(y)$ for all $y \in \mathcal{D}_i(X)$, $X \in \mathbb{B}$ not only for branches but for arbitrary

pairs of paths $p = (i \rightarrow \dots \rightarrow j)$, $p' = (i \rightarrow \dots \rightarrow j')$ starting in a common branching index $i \in Br(\mathbb{I})$ but not necessarily ending at a minimal index.

Theorem 6.9 (Different Paths Connecting Affected Minimal Components). *Let $\mathbb{G} = Set^{\mathbb{B}}$ and $\mathcal{D} : \mathbb{I} \rightarrow \mathbb{G}$ be a diagram with \mathbb{I} a finite directed multigraph without directed cycles. Let*

$$\mathcal{D} \xrightarrow{\kappa} \Delta S$$

be a colimiting cocone with image-disjoint \mathcal{D} . The following are equivalent:

- (1) *The cocone has the Van Kampen property*
- (2) $\forall X \in \mathbb{B}, i, j \in Af(\mathbb{I}), z \in \mathcal{D}_i(X), z' \in \mathcal{D}_j(X)$: *There are no two different proper paths from z to z'*
- (3) $\forall X \in \mathbb{B}, i, j \in Af(\mathbb{I}), z \in \mathcal{D}_i(X), z' \in \mathcal{D}_j(X)$: *There are no two different inner-cycle free proper paths from z to z'*

Proof. The proof of Theorem 6.9 is rather elementary and is given in [12]. □

According to this theorem and according to (6.2), it is only necessary for an algorithm to iterate over branching components and compute paths into potentially affected minimal components. For instance, one has to consider only the small components $\mathcal{D}_{12}, \mathcal{D}_{13}, \mathcal{D}_{23}$ in Fig.1. Again the implementation is independent of whether it ignores inner-cycle free paths or not. It is, however, not guaranteed to find disjoint paths, if VK is violated.

The absence of directed cycles and the presence of image-disjointness are reasonable requirements for many practical use-cases. But circumstances can often be further narrowed. In the rest of this section we consider some other possibly satisfied properties and corresponding alternative checking methods which can simplify VK verification.

Monomorphisms: In some practical cases, the morphisms of \mathcal{D} specify relations between components \mathcal{D}_i and \mathcal{D}_j such that an element in \mathcal{D}_i is related to at most one element of \mathcal{D}_j . In this case all the morphisms \mathcal{D}_d , with d an edge in \mathbb{I} , are monomorphisms. In such a diagram any mapping path P is completely determined by y_0 (or y_n) and the corresponding sequence $[\delta_0, \dots, \delta_{n-1}]$ of edges and opposed edges in \mathbb{I} . Due to Theorem 6.9, the diagram may be not VK only if there are two different sequences of edges and opposed edges between two distinct affected indices in \mathbb{I} . As long as there are no undirected cycles in \mathbb{I} this can not happen, thus the diagram is VK, if there are no undirected cycles in \mathbb{I} .

If there are undirected cycles in \mathbb{I} it is surely not enough to require image-disjointness as defined in Def.6.7, see also Fig.1. Instead, we can ensure VK by the stronger requirement that all the undirected cycles in \mathbb{I} are broken in \mathcal{D} : An undirected cycle of edges in \mathbb{I} is **broken in \mathcal{D}** if for one of the situations

$$\dots \xrightarrow{d_{n-1}} \dots \xrightarrow{d_0} \xleftarrow{d'_0} \dots \xleftarrow{d'_{m-1}} \dots$$

in the edge sequence with $1 \leq n, m$ the morphisms $\mathcal{D}_{d_0} \circ \dots \circ \mathcal{D}_{d_{n-1}}$ and $\mathcal{D}_{d'_0} \circ \dots \circ \mathcal{D}_{d'_{m-1}}$ are image disjoint.

In the example in Figure 1 this condition is not satisfied. In the example "parametrized specification with import", however, it is quite natural that \mathcal{D}_{ib} and \mathcal{D}_r are image disjoint since the "imported component" \mathcal{D}_I is not part of the "parameter component" \mathcal{D}_p .

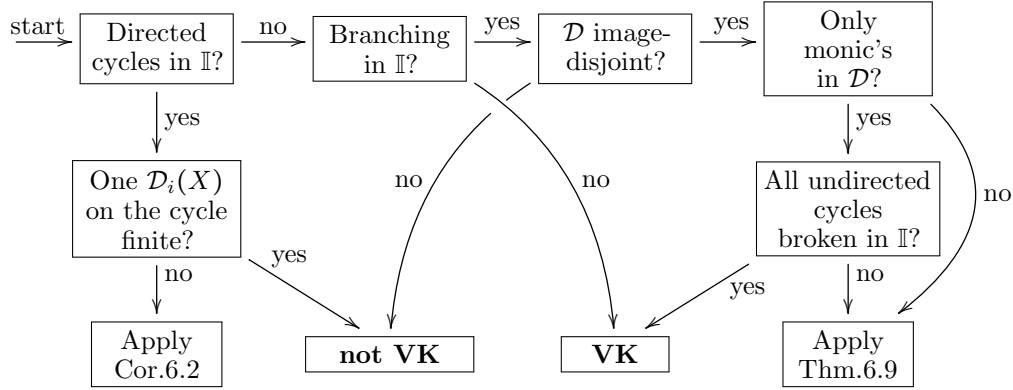


Figure 4: Decision diagram

6.5. Decision Diagram and VK Verification Algorithm. In practical cases, as outlined e.g. in Sect.3.3, colimit computation is obligatory. Verification of the Van Kampen property must follow, if we want to verify compositionality. It would thus be a nice side effect to have a possibility to check VK simultaneously with colimit computation such that there is no increase in time complexity! We will now shortly discuss, that with the results gained so far, this is indeed possible.

For this, let's summarize the outcome of the previous sections as a decision algorithm, see Fig.4. With the exception of rare cases in which there is a directed cycle in \mathbb{I} whose component carrier sets are all infinite, it is possible to easily reach an early decision, if either there are directed cycles in \mathbb{I} or if there is no branching in \mathbb{I} . This analysis is restricted to the small graph \mathbb{I} . In the presence of branching, the natural next step is to check violation of image-disjointness in \mathcal{D} to immediately deduce violation of VK (Fact 6.8). Image-disjointness can immediately be confirmed, if all branches diverge. Otherwise, the mapping behavior along branches has to be investigated, which may be more costly.

Hence the *combined algorithm for colimit computation and Van Kampen verification* comprises the following steps:

- (1) Preprocessing of the data shows whether we are on a decision route in Fig.4 on which Thm.6.9 will be applied. In this case there are *no directed cycles in \mathbb{I}* and \mathcal{D} is *image-disjoint*.
- (2) $vk := true$;
- (3) $\cong_X := \{(z, z) \mid z \in \mathcal{D}_j(X), j \in Min(\mathbb{I})\}$ for all $X \in \mathbb{B}$;
- (4) For each branching component \mathcal{D}_i , each $X \in \mathbb{B}$ and for each $y \in \mathcal{D}_i(X)$, do:
 - (a) Add images (z, z') to \cong_X according to primary identifications in step 2 in the specialized colimit computation.
 - (b) Keep \cong_X transitive by adding all arising transitive pairs from the last enhancement.
 - (c) Whenever in the two previous steps a pair (z, z') is added for the second time, $vk := false$ (cf. Theorem 6.9).
- (5) Compute colimit cocone κ as in (6.1) using the family $\cong = (\cong_X)_{X \in \mathbb{B}}$.
- (6) Return (κ, vk)

7. CONCLUSION AND FUTURE WORK

In general, arbitrary diagrams in arbitrary categories are not VK. Even if we restrict to presheaf topoi, many diagrams are not VK. In the paper we presented a feasible condition (Thm. 3.5) to check if a diagram in a presheaf topos is VK or not.

As suggested by the example in Sect.3.3, modelers may well work with a non-VK-diagram (of software models), if they have a common understanding of the used natural transformation $\tau : \mathcal{E} \Rightarrow \mathcal{D}$, i.e., if they know how to avoid "twisting anomalies" as shown in the example. Hence, the natural next step will be to look for feasible conditions that a given $\tau : \mathcal{E} \Rightarrow \mathcal{D}$ is in the image of κ^* , even if the diagram is not VK. We may allow non-uniqueness of mapping paths in diagrams of models, but then paths in the diagram of instances have to be exact copies of them, i.e., path liftings from models to instances must behave like discrete fibrations. It is worth to underline that the instances we get from a given "indexed semantics" via a corresponding variant of the Grothendieck construction [28] are always contained in the image of κ^* up to isomorphism.

An interesting research direction arises from counter-examples in Sect. 5. It seems to be easy to find categories, where necessity of the Van Kampen property is violated although path uniqueness holds. Although being artificial and practically not relevant, the example showing violation of path uniqueness despite validity of VK is interesting: there do not seem to be other substantially different examples of this type. Is it possible to have violation of path-uniqueness and still validity of the Van Kampen property in more practical examples? We conjecture that the implication "VK \Rightarrow Path-Uniqueness" is very natural and holds in a wider variety of (set-based) categories.

For topologists being familiar with homotopy theory [20], violation of path uniqueness strongly resembles (continuous) paths that can not be contracted to a point. Hence, an interesting further research direction is to investigate, how path lifting (e.g. of covering morphisms) is connected with our investigations. Moreover, discrete unique path lifting is discussed in the theory of (split) fibrations [25]. The ultimate goal, however, is to find a categorical counterpart for the path-uniqueness criterion (Theorem 3.5), which states a necessary and sufficient condition for validity of the Van Kampen property in more general categories. Is such a condition significantly different from the bilimit condition mentioned in the introduction and the universal property in [7] and can we benefit from results of higher order category theory, e.g. [17]?

REFERENCES

- [1] M. Bunge and S. Lack. Van Kampen Theorems for Topoi. *Advances in Mathematics*, 179:291 – 317, 2003.
- [2] Z. Diskin and U. Wolter. A Diagrammatic Logic for Object-Oriented Visual Modeling. *Electr. Notes Theor. Comput. Sci.*, 203(6):19–41, 2008. doi:10.1016/j.entcs.2008.10.041.
- [3] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformations*. Springer, 2006.
- [4] Hartmut Ehrig, M. Grosse-Rhode, and U. Wolter. Applications of Category Theory to the Area of Algebraic Specification in Computer Science. *Applied Categorical Structures*, 6:1–35, 1998.
- [5] Jose Luiz Fiadeiro. *Categories for Software Engineering*. Springer, 2005.
- [6] Robert Goldblatt. *Topoi: The Categorical Analysis of Logic*. Dover Publications, 1984.
- [7] T. Heindel and P. Sobociński. Van Kampen Colimits as Bicolimits in Span. In A. Kurz, M. Lenisa, and A. Tarlecki, editors, *Algebra and Coalgebra in Computer Science*, volume 5728 of *Lecture Notes in Comput. Sci.*, pages 335–349. Springer Berlin / Heidelberg, 2009. doi:10.1007/978-3-642-03741-2_23.

- [8] G Janelidze and W. Tholen. Facets of Descent, I. *Appl. Categorical Structures*, 2:245–281, 1994. doi:10.1007/BF00878100.
- [9] Wolfram Kahl. Collagories: Relation-algebraic Reasoning for Gluing Constructions. *J. Log. Algebr. Program.*, 80(6):297–338, 2011. doi:10.1016/j.jlap.2011.04.006.
- [10] Wolfram Kahl. *Categories of Coalgebras with Monadic Homomorphisms*, pages 151–167. Springer, Berlin, Heidelberg, 2014. doi:10.1007/978-3-662-44124-4_9.
- [11] Harald König, Michael Löwe, Christoph Schulz, and Uwe Wolter. Van Kampen Squares for Graph Transformation. In *Graph Transformation - 7th International Conference, ICGT 2014, Held as Part of STAF 2014, York, UK, July 22-24, 2014. Proceedings*, pages 222–236, 2014. doi:10.1007/978-3-319-09108-2_15.
- [12] Harald König and U. Wolter. Van Kampen Colimits in Presheaf Topoi. Technical report, University of Applied Sciences, FHDW Hannover, 2016. URL: <http://fhdwdev.ha.bib.de/public/papers/02016-02.pdf>.
- [13] Harald König and U. Wolter. Being Van Kampen is a Uniqueness Property in Presheaf Topoi. CALCO 2017 Pre-Proceedings. URL: <http://coalg.org/mfps-calco2017/calco-papers/calco2017-16.pdf>
- [14] S. Lack and P. Sobociński. Adhesive Categories. In *Foundations of Software Science and Computation Structures (FoSSaCS '04)*, volume 2987, pages 273–288. Springer, 2004. doi:10.1007/978-3-540-24727-2_20.
- [15] S. Lack and P. Sobociński. Toposes are Adhesive. *Lecture Notes in Comput. Sci.*, 4178:184–198, 2006. doi:10.1007/11841883_14.
- [16] Michael Löwe. Van Kampen Pushouts for Sets and Graphs. Technical report, University of Applied Sciences, FHDW Hannover, 2010.
- [17] Jacob Lurie. Higher Algebra. <http://www.math.harvard.edu/~lurie/papers/HA.pdf>, Sept. 2017.
- [18] Moerdijk I. Mac Lane, S. *Sheaves in Geometry and Logic. A first introduction to topos theory*. Springer, 1992.
- [19] Saunders Mac Lane. *Categories for the Working Mathematician, Second edition*. Springer, 1998.
- [20] J.P. May. *A Concise Course in Algebraic Topology*. Chicago Lectures in Mathematics. The University of Chicago Press, 1999. URL: <http://dx.doi.org/10.1007/978-3-642-17336-3>.
- [21] Mehrdad Sabetzadeh, Shiva Nejati, Sotirios Liaskos, Steve M. Easterbrook, and Marsha Chechik. Consistency Checking of Conceptual Models via Model Merging. In *Requirements Engineering Conference*, pages 221–230, 2007.
- [22] Donald Sannella and Andrzej Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2012. URL: <http://dx.doi.org/10.1007/978-3-642-17336-3>, doi:10.1007/978-3-642-17336-3.
- [23] Herbert Seifert. Konstruktion dreidimensionaler geschlossener Räume. *Dissertation, University of Dresden*, 1931.
- [24] P. Sobociński. Deriving Process Congruences from Reaction Rules. Technical Report DS-04-6, BRICS Dissertation Series, 2004.
- [25] T. Streicher. Fibred Categories à la Jean Bénabou <https://arxiv.org/pdf/1801.02927v2.pdf>, 2004.
- [26] E. R. van Kampen. On the Connection between the Fundamental Groups of some Related Spaces. *American Journal of Mathematics*, 55:261 – 267, 1933.
- [27] A. Vistoli. Grothendieck Topologies, Fibered Categories and Descent Theory. *Fundamental Algebraic Geometry, Math. Surveys Monogr., Amer. Math. Soc., Providence, RI, 2005*, 123:1 – 104, 2005.
- [28] U. Wolter and Z. Diskin. From Indexed to Fibred Semantics – The Generalized Sketch File –. Reports in Informatics 361, Dep. of Informatics, University of Bergen, 2007.
- [29] U. Wolter and H. König. Fibred Amalgamation, Descent Data, and Van Kampen Squares in Topoi. *Applied Categorical Structures*, 23(3):447 – 486, 2015. doi:10.1007/s10485-013-9339-2.
- [30] H. Reichel. Initial Computability, Algebraic Specifications, and Partial Algebras. *Oxford University Press* 1987
- [31] U. Wolter. An Algebraic Approach to Deduction in Equational Partial Horn Theories. *J. Inf. Process. Cybern. EIK* 27(2): 85 – 128, 1990
- [32] U. Wolter, M. Klar, R. Wessäly, F. Cornelius. Four Institutions – A Unified Presentation of Logical Systems for Specification. *TU Berlin, Fachbereich Informatik*, 1994, 94-24