

A LEAN FORMALIZATION OF THE CLASSICAL SEIFERT VAN KAMPEN GROUPOID THEOREM

ARTHUR FREITAS RAMOS, DAVID BARROS HULAK, AND RUY J. G. B. DE QUEIROZ

ABSTRACT. We explain a Lean 4 formalization of the classical Seifert van Kampen theorem for the full fundamental groupoid. For an arbitrary topological space covered by two open subsets, the inclusion-induced square of ordinary continuous-path fundamental groupoids is a pushout in the category of categories. Every point is retained as an object; no basepoint, connectedness, or separation hypothesis is required. The development identifies an auxiliary directed path category for the indiscrete preorder with Mathlib’s fundamental groupoid through strictly inverse functors and naturality equalities. It then assembles the pushout universal property from attributed path-subdivision and homotopy-grid helpers adapted from the directed-topology formalization of Basold, Bruin, and Lawson. We describe the descent construction, its transfer to ordinary fundamental groupoids, and the statement and verification boundaries of the exact artifact registered in Palomar. This is an expository account of a classical theorem and an existing formal artifact, with no claim of mathematical novelty or formalization priority.

1. SCOPE AND MATHEMATICAL CONTEXT

The Seifert van Kampen theorem expresses a local-to-global principle for paths: information on two open subsets and their overlap determines the fundamental groupoid of their union. The groupoid formulation keeps all endpoints of paths available. It therefore accommodates disconnected intersections without selecting one point that could fail to represent an entire component. Brown’s Theorem 3.4, with its full-groupoid case in Section 5, is the classical mathematical source relevant here [2]. The selected formal statement is the two-open-set, all-object version, rather than Brown’s more general treatment of chosen sets of objects.

This note explains the Lean artifact registered as PALOMAR-2026-09-25-000004, version 1 [5]. All implementation claims refer to the registered commit `874680e16db88b4a41daadf56eafbac79fd2f748` of *classical-svk-lean* [6]. The public record, rather than a moving branch or an older candidate named in a README, determines this pin. Its theorem is `ClassicalSVK.seifert_van_kampen_groupoid` in `Solution.lean`.

The substantive proof infrastructure is shared with a prior formalization of directed topology. Basold, Bruin, and Lawson formalized a directed van Kampen theorem in Lean [1]. The present artifact specializes the directedness relation to the indiscrete preorder, reuses attributed subdivision and homotopy-grid helpers, and transfers a directly assembled universal property to Mathlib’s ordinary fundamental groupoid. The selected proof does not call the packaged directed van Kampen theorem. This distinction identifies the actual dependency; it does not make the inherited descent construction an independently authored proof.

Date: September 30, 2026.

2020 Mathematics Subject Classification. 55Q05, 18B40, 68V20.

Key words and phrases. Seifert van Kampen theorem, fundamental groupoid, open cover, categorical pushout, Lean 4, Mathlib.

Author ORCID identifiers: Arthur Freitas Ramos, 0009-0003-3568-0325; David Barros Hulak, 0009-0002-8056-1774; Ruy J. G. B. de Queiroz, 0000-0003-1482-0977.

Copyright 2026 the authors. Licensed under Creative Commons Attribution 4.0 International (CC BY 4.0).

Our purpose is to connect the mathematical statement with the source interfaces and recorded verification. We claim neither a new topological theorem nor priority for a formalization. The discussion below distinguishes proved declarations from explanatory mathematical notation and from historical verification reports.

2. THE ORDINARY FUNDAMENTAL GROUPOID AND THE EXACT STATEMENT

2.1. Objects and path classes. Let X be a topological space. A path from x to y is a continuous map $p: [0, 1] \rightarrow X$ with $p(0) = x$ and $p(1) = y$. Two such paths represent the same morphism when there is a homotopy $H: [0, 1] \times [0, 1] \rightarrow X$ between them that fixes both endpoints throughout. The fundamental groupoid $\Pi_1(X)$ has the points of X as objects and these path classes as morphisms. Constant paths supply identities, concatenation supplies composition, and path reversal supplies inverses.

The implementation uses Mathlib's `FundamentalGroupoid`, `Path`, and `Path.Homotopic.Quotient` [4]. A continuous map $f: X \rightarrow Y$ induces a functor $\Pi_1(f)$ by mapping points and postcomposing path representatives. The groupoid functor is `FundamentalGroupoid.fundamentalGroupoidFunctor`. Thus the theorem concerns actual continuous paths in topological subspaces. It does not replace them by a syntactic computational-path presentation.

Subsets in the statement carry their subspace topologies. In Lean, U is the subtype $\{x : X \mid x \in U\}$, while $U \cap V$ is the subtype with both membership proofs. Let

$$i_U: U \cap V \rightarrow U, \quad i_V: U \cap V \rightarrow V, \quad j_U: U \rightarrow X, \quad j_V: V \rightarrow X$$

be the evident inclusions. The source constructs these as continuous maps in `TopCat`; the two composites from $U \cap V$ to X agree.

Theorem 2.1 (Registered classical groupoid statement). *For every topological space X and open subsets $U, V \subseteq X$ with $U \cup V = X$, the square*

$$(1) \quad \begin{array}{ccc} \Pi_1(U \cap V) & \xrightarrow{\Pi_1(i_U)} & \Pi_1(U) \\ \downarrow \Pi_1(i_V) & & \downarrow \Pi_1(j_U) \\ \Pi_1(V) & \xrightarrow{\Pi_1(j_V)} & \Pi_1(X) \end{array}$$

*is a pushout in **Cat**, after viewing each groupoid as a category.*

This is the mathematical reading of `ClassicalSVK.completeStatement`. Its quantified context supplies a topological-space instance and the hypotheses that U and V are open and $U \cup V$ is the universal set. Lean encodes these using `TopologicalSpace`, `IsOpen`, and `Set.univ`. There is no chosen basepoint, nonemptiness, path-connectedness, Hausdorff condition, or manifold assumption. The conclusion is Mathlib's `IsPushout`, applied after `CategoryTheory.GrpD.forgetToCat`. The theorem is therefore stated in the ordinary category of categories and functors, not just inside the category of groupoids and not as a homotopy pushout.

2.2. The strict universal property. Write I_U, I_V, J_U, J_V for the four functors in (1). For a target category C in the relevant universe, suppose that functors

$$F_U: \Pi_1(U) \rightarrow C, \quad F_V: \Pi_1(V) \rightarrow C$$

satisfy the equality

$$(2) \quad F_U \circ I_U = F_V \circ I_V.$$

The pushout assertion says that there is a unique functor $F: \Pi_1(X) \rightarrow C$ with

$$(3) \quad F \circ J_U = F_U, \quad F \circ J_V = F_V.$$

These are equalities of functors, including their actions on objects. The statement does not start with a chosen natural isomorphism between the restrictions. A bicategorical gluing problem with isomorphism-valued compatibility would need a different formulation.

This strictness is useful for understanding the implementation. At a point of the overlap, (2) equates the two object values, so the descent construction can transport morphisms between literally equal target objects. Its endpoint casts are proof bookkeeping for these identifications, not extra path-connectedness assumptions.

Remark 2.2 (Based groups and disconnected overlaps). For $x \in X$, the usual fundamental group $\pi_1(X, x)$ is the automorphism group of x in $\Pi_1(X)$. Retaining all objects allows the statement to cover a disconnected overlap, whose different components can contribute different connecting paths. The familiar based amalgamated-product theorem requires its own hypotheses and comparison argument. Taking an automorphism group at one object is not, by itself, a pushout-preserving operation. The registered declaration proves the full-groupoid pushout, not a separate based-group presentation or an explicit computation for a particular space.

3. THE INDISCRETE PREORDER BRIDGE

3.1. Why every ordinary path becomes directed. The helper library describes directed spaces and directed path classes. To use it for ordinary topology, the bridge equips X with the preorder

$$(4) \quad x \preceq y \iff \text{True}.$$

This is `ClassicalSVK.universalPreorder`. The word *indiscrete* here describes the order relation, not the topology of X . The original topology remains unchanged. Since every pair of points is ordered, all monotonicity conditions into X are automatic.

The map `pathToDipath` therefore takes any ordinary `Path x y` to a `Dipath x y` with the same underlying continuous map. Forgetting directedness recovers the original path by `pathToDipath_toPath`. The same relation is used on the relevant subtypes, so this observation applies to U , V , and $U \cap V$ as well. No local connectivity argument is involved.

There are also two implications at the level of homotopies. The declaration `dipathDihomotopic_to_pathHomotopic` sends directed path equivalence to ordinary endpoint-preserving homotopy. The source's directed equivalence is the equivalence closure of directed homotopies, so the proof handles reflexivity, symmetry, and transitivity. In the other direction, `pathHomotopic_to_dipathDihomotopic` promotes an ordinary homotopy: the extra directedness requirement is again automatic under (4). These proofs justify lifting the maps to quotients.

3.2. Strictly inverse functors. Let $D(X)$ denote the auxiliary fundamental category for the indiscrete preorder, and let $P(X) = \Pi_1(X)$ denote the ordinary groupoid viewed as a category. The bridge defines

$$R_X: D(X) \rightarrow P(X), \quad S_X: P(X) \rightarrow D(X).$$

Both functors preserve the underlying point. Their actions on morphisms are the quotient lifts `directedClassToPathClass` and `pathClassToDirectedClass`. Quotient induction reduces inverse identities to the corresponding equalities of path representatives. The functors also preserve identities and composition.

The source proves the equalities

$$(5) \quad S_X \circ R_X = \text{id}_{D(X)}, \quad R_X \circ S_X = \text{id}_{P(X)}.$$

They are recorded in `Bridge/Iso.lean` as `directedToClassical_comp_classicalToDirected` and `classicalToDirected_comp_directedToClassical`. Thus the bridge gives an isomorphism of categories in **Cat**, stronger than merely specifying inverse functors up to natural isomorphism. It does not assert that the two Lean types are definitionally identical.

For an inclusion $f: A \rightarrow B$, let $D(f)$ and $P(f)$ be its induced functors. The bridge additionally proves

$$(6) \quad R_B \circ D(f) = P(f) \circ R_A, \quad S_B \circ P(f) = D(f) \circ S_A.$$

These are `directedToClassical_naturality` and `classicalToDirected_naturality`, in the two naturality modules. In the selected proof they are instantiated for all four inclusions. These equalities ensure that the path-class identification respects the whole cover diagram, not just each of its four categories in isolation.

4. DESCENT FROM PATH SUBDIVISION AND HOMOTOPY GRIDS

The central helper module is `Lean4/path_descent_helpers.lean`. It is extracted from the pinned directed-topology source's `Lean4/directed_van_kampen.lean`. The constructions below describe its specialization to (4); the helper code itself is attributed infrastructure, not a new construction of this note.

4.1. Objects and covered paths. Take compatible functors from $D(U)$ and $D(V)$ to C , denoted by G_U and G_V . On a point $x \in X$, the helper `FunctorOnObj` uses $G_U(x)$ if $x \in U$ and $G_V(x)$ otherwise. The cover supplies membership in V in the second case. If a point lies in both sets, compatibility gives equality of the two possible object values. The declarations `functorOnObj_apply_one` and `functorOnObj_apply_two` expose these identifications.

A path is *covered* when its entire image lies in one cover member. For such a path, restrict it to the corresponding subtype, apply the appropriate functor, and transport its source and target to the chosen object values. The helper `FunctorOnHomOfCovered` implements this operation. When the image lies in both members, the path is a path in $U \cap V$, and compatibility proves the two assignments equal through `functorOnHomOfCoveredAux_equal`. This is the first local gluing step.

4.2. An arbitrary path and independence of subdivision. For a continuous path $p: [0, 1] \rightarrow X$, the preimages of U and V form an open cover of the compact unit interval. A finite subdivision can be chosen so that every subpath is covered. The formal infrastructure uses equal subdivisions indexed by a natural number: `Dipath.covered_partwise.has_subpaths` in `Lean4/path_cover.lean` proves existence of the needed `covered_partwise` data.

If $p_1, \dots, p_N$ are consecutive reparametrized subpaths, the proposed value is the composite of their covered-path values:

$$(7) \quad L([p]) = L_0([p_N]) \circ \dots \circ L_0([p_1]).$$

Here L_0 denotes the local assignment, and our conventional right-to-left composition agrees with the chronological order of the path pieces. The source writes this as a left-to-right categorical composite using Lean's category notation. Endpoint casts align adjacent morphisms.

Two points must be checked before (7) defines descent. First, the value of a covered path must agree with the value obtained by splitting it. This uses the functor law on each cover member and endpoint-preserving reparametrization. Second, different permitted subdivision sizes must give the same composite. The helper refines subdivisions and compares them at a common size. In its indexing, n describes $n + 1$ pieces, so the common refinement size is governed by the product of the piece counts. The key declarations are `functorOnHomOfCoveredPartwise_refine` and `functorOnHomOfCoveredPartwise_unique`.

The assignment `FunctorOnHomAux` chooses one subdivision using `Classical.choose`. Its subsequent theorem `functorOnHomAux_apply` allows any valid subdivision to compute the same value. Hence the chosen data do not affect the resulting morphism. Further lemmas establish the constant-path and concatenation laws. This use of classical choice concerns selecting finite witnesses; the artifact does not provide an executable algorithm that computes fundamental groupoids from arbitrary topological spaces.

4.3. Homotopy invariance. A path assignment must also descend through the homotopy quotient. For an endpoint-preserving homotopy $H: [0, 1] \times [0, 1] \rightarrow X$, the preimages of U and V cover the compact square. A sufficiently fine rectangular grid has every rectangle mapped into one member. The existence result is `DirectedMap.Dihomotopy.coveredPartwise_exists` in `Lean4/dihomotopy_cover.lean`.

Inside a covered rectangle, the two routes along its boundary represent homotopic paths in the same subspace, so the local functor equates their values. The helper combines these equalities across columns and rows. Its proof handles intermediate paths whose endpoints vary, and then specializes to the endpoint-preserving case. The fixed exterior sides contribute identities, yielding equality of the values of the original paths. The formal assembly includes `functorOnHomAux_of_partwise_covered_dihomotopic` and finally `functorOnHomAux_of_dihomotopic`; the latter respects the full symmetric equivalence closure of directed homotopy.

The quotient lift `FunctorOnHom` is now well defined. Identity and composition lemmas combine it with `FunctorOnObj` to form `DirectedVanKampen.PushoutFunctor.Functor`, which we denote by $L: D(X) \rightarrow C$. The factorization lemmas `functor_comp_left` and `functor_comp_right` show that L restricts exactly to the initial functors. The lemma `functor_uniq` compares objects using the cover and compares morphisms using covered pieces of a subdivision. It proves equality of any other extending functor with L .

5. ASSEMBLING AND TRANSFERRING THE PUSHOUT

In `ClassicalSVK/Pushout.lean`, the declaration `ClassicalSVK.classicalOpenCoverPushout` first constructs the auxiliary inclusion square. It uses `PushoutAlternative.isPushout_alternative`, whose hypothesis is the strict unique-extension property for compatible functors. The proof supplies the descent functor and the three factorization and uniqueness lemmas just described. This produces a pushout for $D(U \cap V)$, $D(U)$, $D(V)$, and $D(X)$.

The ordinary pushout is then built explicitly through the bridge. To see the transfer, start with a compatible pair F_U, F_V for $P(U), P(V)$. Define

$$G_U = F_U \circ R_U, \quad G_V = F_V \circ R_V.$$

Naturality (6) transports their compatibility to the auxiliary overlap. Auxiliary descent provides $L: D(X) \rightarrow C$. Set

$$(8) \quad F = L \circ S_X: P(X) \rightarrow C.$$

For example,

$$F \circ P(j_U) = L \circ S_X \circ P(j_U) = L \circ D(j_U) \circ S_U = G_U \circ S_U = F_U,$$

using naturality, auxiliary factorization, and $R_U \circ S_U = \text{id}_{P(U)}$. The argument for V is the same.

If F' is another ordinary extension, then $F' \circ R_X$ extends G_U, G_V by naturality. Auxiliary uniqueness forces $F' \circ R_X = L$. Composing with S_X and using $R_X \circ S_X = \text{id}_{P(X)}$ gives $F' = F$. This also explains why strict inverse identities are especially convenient: the target of the proof is a strict pushout in **Cat**.

In the source, `toDirectedCocone` transports an ordinary pushout cocone, `liftDesc` gives its mediating morphism, and `PushoutCocone.IsColimit.mk` packages the factorization and uniqueness proofs. The final use of `IsPushout.of_isColimit` exposes the claimed square. `Solution.lean` introduces the quantified space, sets, and open-cover hypotheses and applies this result. These constructions, rather than an invocation of `DirectedVanKampen.directed_van_kampen`, are the route followed by the selected theorem.

6. STATEMENT BOUNDARY AND RECORDED VERIFICATION

6.1. A challenge is not an implementation proof. The repository separates a closed statement from its proof. `Challenge.lean` imports only the following Mathlib modules:

- `Mathlib.AlgebraicTopology.FundamentalGroupoid.Basic`;
- `Mathlib.CategoryTheory.CommSq`;
- `Mathlib.CategoryTheory.Limits.Shapes.Pullback.IsPullback.Basic`.

It defines `completeStatement` and gives the selected theorem one intentional `sorry` placeholder. This is a comparator challenge, not a proof. `Solution.lean` does not import that module; it repeats the same statement and proves it through `ClassicalSVK.Pushout`. The selected implementation and its project-local import closure contain no `sorry`, `admit`, or new axiom declaration in the source inspected for this note.

The comparator configuration selects both the theorem `ClassicalSVK.seifert_van_kampen_groupoid` and the definition `ClassicalSVK.completeStatement`. The independent statement copy helps keep the mathematical specification readable without importing the auxiliary directed infrastructure into the challenge. Comparing the statement is important because a proof of a changed proposition would not establish the advertised theorem.

6.2. Version pins and evidence. The registered artifact uses Lean `leanprover/lean4: v4.35.0-rc2` and Mathlib commit `065356127b1dc0016f66b7283ce0ce2c4055aa55`. The directed-topology source baseline is commit `009529606c66d37ef93b4b81b8587f71ce4d2c56` [3]. Its copied module tree is compatibility-ported into the repository; it is not a second unpinned Lake dependency. `Lean4/vendor-manifest.json` records source and port hashes, while `Lean4/PORTING.md` records 26 compatibility-ported files and the extracted descent helper.

The immutable Palomar record [5] reports registration on September 25, 2026, for the source commit used here. It records the selected theorem and statement, the toolchain and dependency pins, the permitted axioms `propext`, `Classical.choice`, and `Quot.sound`, and mechanical verification on September 24, 2026. Its verification section identifies the workflow run and named kernels `nanoda` and `con-ron`. These are recorded artifact-verification facts; they are not a fresh Lean compilation performed for this manuscript.

The source provides reproducibility commands for building the challenge and solution, auditing the compiled closed statement, checking axioms and package structure, and reconciling provenance. In order, they are:

```
lake build Challenge Solution
lake env lean scripts/check-closed-statement.lean
python scripts/check-axioms.py
python scripts/check-package.py
python scripts/check-provenance.py
```

The axiom script queries the selected theorem through Lean and checks its exact dependency set. The provenance script checks both immutable upstream hashes and the selected import graph, including exclusion of the packaged directed theorem. Reading these scripts explains their intended checks; it does not itself execute them.

Preparation of this note involved source and registry inspection and compilation of the TeX manuscript. We do not claim a new end-to-end Lean build or an independent human line-by-line review. The pinned repository’s README and verification prose contain pre-registration status statements. For present registration status, the later immutable public registry record is the relevant evidence. Registry registration and its reported checks do not certify the prose exposition, mathematical novelty, or journal peer review.

7. PROVENANCE AND RELATED FORMALIZATIONS

Brown is credited for the classical groupoid theorem. Basold, Bruin, and Lawson are credited for the directed-topology formalization and the subdivision and homotopy-grid infrastructure used here. Mathlib supplies the ordinary path groupoid, continuous maps, subspaces, and categorical pushout APIs. The bridge and explicit transfer identify how these source components meet. The selected theorem is a specialization and integration of attributed prior work, not an assertion of independent authorship of its helper lemmas.

The repository also discloses earlier computational-path developments at commit `257c659b7973aeda900d86a5da73b208712c7523` [7]. Its provenance record reports no reuse of

source files or proof terms from that project. The formal objects in the registered theorem are the ordinary continuous-path groupoids of actual subspaces. This difference in model should be preserved when comparing artifacts; a theorem about a symbolic presentation is not automatically the same formal statement.

A related Isabelle/HOL entry by Ramos, Hulak, and de Queiroz [8] treats a based fundamental-group theorem, with a point in the overlap and a path-connected intersection. Its published description states a bijection with a carrier-based amalgamated free product and an encode/decode proof architecture. The present repository reports no reuse of Isabelle source or proof. We make no formal cross-system equivalence claim and no assertion that one artifact subsumes every result of the other.

The registered Lean project’s metadata identifies Arthur Freitas Ramos as its author and responsible maintainer. The three names on this note are the manuscript authors; that authorship list does not retroactively change the source project’s recorded authorship. Its first-party code is Apache-2.0 licensed, the vendored directed-topology code retains the MIT notice in `Lean4/LICENSE.md`, and Mathlib retains its own licensing. The CC BY 4.0 license on this manuscript and its TeX source does not relicense those dependencies.

8. CONCLUDING SCOPE AND DISCLOSURE

The main interface is a strict universal property: compatible functors on the two ordinary fundamental groupoids extend uniquely to the whole space. The proof engineering makes three boundaries explicit. Homotopy classes must be preserved when moving between path models; the bridge must respect the inclusion diagram; and the chosen finite subdivisions must yield a value independent of choices and representatives. These are the points at which the source connects classical topological descent to the exact Mathlib statement.

The artifact covers arbitrary two-open-set covers, including disconnected or empty overlaps. It does not separately formalize the general arbitrary open-cover theorem, a chosen-basepoint presentation, a bicategorical pushout, higher homotopy excision, or particular fundamental-group computations. Such directions would require additional statements and proofs beyond the registered declaration.

AI assistance and human understanding. This manuscript was drafted primarily with GPT-6.1 assistance, including exposition, source alignment, and typesetting. The pinned repository’s metadata separately reports GPT-6-Astra assistance for proof engineering, provenance review, and package preparation. Some human understanding informed the work; no claim is made that every listed author independently understands every formal proof step. The model-assisted review of this manuscript is not independent human peer review, and archive acceptance would not establish such review.

Availability. The complete Lean development, dependency and provenance records, and verification scripts are available at the pinned repository [6]; the registered public record is [5]. The manuscript source archive contains only `main.tex` and `references.bib`. The Lean code remains in its separately licensed source repository.

REFERENCES

1. Henning Basold, Peter Bruin, and Dominique Lawson, *The directed Van Kampen theorem in Lean*, 15th International Conference on Interactive Theorem Proving (ITP 2024) (Yves Bertot, Temur Kutsia, and Michael Norrish, eds.), Leibniz International Proceedings in Informatics, vol. 309, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024, <https://doi.org/10.4230/LIPIcs.ITP.2024.8>, pp. 8:1–8:18.
2. Ronald Brown, *Groupoids and Van Kampen’s theorem*, Proceedings of the London Mathematical Society (3) **17** (1967), no. 3, 385–401, <https://doi.org/10.1112/plms/s3-17.3.385>.
3. Dominique Lawson, *Directed-topology-Lean-4*, Source repository, pinned commit 009529606c66d37ef93b4b81b8587f71ce4d2c56, 2024, Attribution and baseline specified by the registered artifact’s vendor manifest. <https://github.com/Dominique-Lawson/Directed-Topology-Lean-4/tree/009529606c66d37ef93b4b81b8587f71ce4d2c56>. Accessed September 30, 2026.

4. Mathlib Contributors, *Mathlib fundamental groupoid and categorical infrastructure*, Source repository, commit 065356127b1dc0016f66b7283ce0ce2c4055aa55, 2026, <https://github.com/leanprover-community/mathlib4/tree/065356127b1dc0016f66b7283ce0ce2c4055aa55>. In particular, `Mathlib/AlgebraicTopology/FundamentalGroupoid/Basic.lean`. Accessed September 30, 2026.
5. Palomar Registry, *Classical Seifert–van Kampen theorem for the fundamental groupoid*, Immutable registry record PALOMAR-2026-09-25-000004, version 1, 2026, Registered September 25, 2026. <https://data.palomar-registry.org/entries/PALOMAR-2026-09-25-000004-v1.json>. Accessed September 30, 2026.
6. Arthur Freitas Ramos, *Classical Seifert–van Kampen theorem for the fundamental groupoid*, Lean repository, registered commit 874680e16db88b4a41daadf56eafbac79fd2f748, 2026, <https://github.com/Arthur742Ramos/classical-svk-lean/tree/874680e16db88b4a41daadf56eafbac79fd2f748>. Accessed September 30, 2026.
7. ———, *ComputationalPathsLean*, Related source repository, snapshot 257c659b7973aeda900d86a5da73b208712c7523, 2026, <https://github.com/Arthur742Ramos/ComputationalPathsLean/tree/257c659b7973aeda900d86a5da73b208712c7523>. Relationship described in the registered artifact’s provenance record. Accessed September 30, 2026.
8. Arthur Freitas Ramos, David Barros Hulak, and Ruy Jose Guerra Barretto de Queiroz, *The classical Seifert–van Kampen theorem*, Archive of Formal Proofs, May 2026, Entry dated May 9, 2026. <https://isa-afp.org/entries/Seifert-Van-Kampen.html>. Accessed September 30, 2026.