Computational Paths in Lean: From Rewrite Equality to Fundamental Groups

Arthur F. Ramos Tiago M. L. de Veras Ruy J. G. B. de Queiroz
Anjolina G. de Oliveira

November 2025

Abstract

We report on the formalisation, in Lean 4, of the theory of computational paths that stems from the labelled natural deduction account of propositional equality. Building on the rewrite reasoning developed in [**?**], we implemented an explicit calculus of paths, rewrite sequences, quotients, and higher groupoid structure that scales to homotopical applications. The resulting library includes normalisation and confluence witnesses, loop group constructions, encode–decode proofs of $\pi_1(\mathbb{S}^1) \cong \mathbb{Z}$ and $\pi_1(\mathbb{T}^2) \cong \mathbb{Z} \times \mathbb{Z}$, and automation that internalises the derivations used in labelled natural deduction [**?**]. Besides reviewing the mathematical background, we describe the Lean architecture, the rewrite certification pipeline, and the interaction between computational paths, homotopy type theory, and higher inductive types. The paper closes with a roadmap for extending the framework to new higher-inductive signatures, higher-genus surfaces, and cubical models.

Contents

1 Introduction

Propositional equality in intensional Martin-Löf type theory is famously subtle: the identity type $Id_A(a, b)$ carries more structure than mere reflexivity, and its inhabitants admit distinct canonical forms. Starting from Hofmann and Streicher's groupoid interpretation [**?**], and reinforced by the homotopical perspective advocated by Awodey [**?**] and the Univalent Foundations programme [**?**], we now look at proofs of equality as paths. The

computational path programme initiated by de Queiroz, de Oliveira, and collaborators takes this idea seriously: instead of postulating that reflexivity alone generates Id-inhabitants, the rules explicitly mention *sequences of rewrites* and attach identifiers to them, so that the system can talk about equality between equalities [**?**].

The present Lean development pursues two complementary goals. First, we turn the labelled deduction system—known as LNDEQ—into a working calculus that can be executed, inspected, and extended inside a proof assistant. Secondly, we push that calculus far enough to reprove landmark results from homotopy type theory (HoTT), notably the computation of $\pi_1(\mathbb{S}^1)$ and $\pi_1(\mathbb{T}^2)$ via encode–decode arguments, and to align it with the labelled natural deduction approach to fundamental groups developed in [**?**]. The net result is a corpus of 12,800+ lines of Lean code (29 modules) that combines rewrite automation, quotient constructions, and homotopical reasoning with a single unifying notion: computational paths. The complete development is available as the open-source `ComputationalPathsLean` repository [**?**].

1.0.0.1 Contributions.

- A Lean-native model of computational paths, covering reflexivity, symmetry, transitivity, substitution, contexts, transport, and dependent congruence, together with rewrite closures Rw and RwEq.
- A catalogue of rewrite rules that mirrors Definition 3.21 of [**?**], plus the confluence witnesses, recursive-path ordering arguments, and automation required to reduce arbitrary derivations to normal form.
- Quotient constructions, loop monoids, loop groups, bicategory and weak 2-groupoid instances, and encode–decode theorems for the circle and the torus.
- A bridge from Lean proofs to the labelled natural deduction methodology that computes fundamental groups of classical surfaces [**?**], showing how the same combinatorial data can be re-used in a proof assistant.

1.0.0.2 Structure of the paper.

Section **??** revisits identity types and computational paths. Section **??** describes the Lean 4 architecture. Section **??** covers homotopical applications. Section **??** connects our development with the labelled deduction calculations of fundamental groups. Section **??** reports on engineering aspects and automation coverage. Sections **??** and **??** discuss related work and future lines of research, followed by concluding remarks in Section **??**.

2 Identity Types and Computational Paths

2.1 Identity types as higher structure

In intensional Martin-Löf type theory, the identity type $Id_A(a, b)$ is generated by the reflexivity constructor $\mathsf{refl}_a : Id_A(a, a)$ and governed by an eliminator J that implements path induction. Hofmann and Streicher showed that the identity type of a type A behaves like the hom-sets of a groupoid whose objects are the elements of A; consequently, identity proofs are generally non-unique [**?**]. Lumsdaine later proved that the tower of iterated identity types carries the structure of a weak ω-groupoid [**?**], and van den Berg and Garner refined the argument in [**?**]. These results legitimise the interpretation of equality proofs as higher-dimensional paths and make space for homotopy-theoretic notions such as composition, inverses, and homotopies between paths.

2.2 Sequences of rewrites

The computational path line of work goes a step further by treating identity proofs as explicit rewrite traces. Instead of deriving equality solely from reflexivity, symmetry, and transitivity, the rules keep track of the concrete rewrites that connect two terms. A judgement

$$s(a, b) : Id_A(a, b)$$

asserts that s is a name for a sequence of rewrites that takes a to b. Reflexivity, symmetry, and transitivity become rewrite combinators, and the equality between equalities can be presented as another rewrite system on rewrite names themselves. Definition 3.21 of [**?**] presents forty canonical rules (augmented to forty-two by the Knuth–Bendix completion) that serve as the axioms of this rewrite layer, commonly referred to as LNDEQ.

In Lean we encode those rewrite rules as inductive constructors over a type family $\mathsf{Path}\, A\, a\, b$ that mirrors the constructors of equality, but never reduces to mere judgemental equality. Each constructor has a ‘Step‘ wrapper, and their transitive closure yields Rw, while a quotient by propositional equality yields RwEq. Automation tactics combine these constructors to discharge obligations about rewrite equivalence.

2.3 LNDEQ and its meta-theory

The rewrite system enjoys normalisation and confluence. On paper, [**?**] argues using recursive-path ordering and a critical-pair analysis. Our Lean library mirrors the argument with three modules:

1. **Instantiation witnesses.** Every rewrite rule is packaged as an enumerated reason together with its parameters. Applying a rule produces an element of ‘Step‘, and the

instantiation record remembers which rule was used.

2. **Termination.** Module `Path/Rewrite/Termination.lean` defines `Termination.Witness`, a record that stores a normal form together with a derivation of a path to that normal form. The recursive-path ordering from [**?**, **?**] is implemented explicitly, showing that each derivation decreases the multiset measure attached to rewrite reasons.

3. **Confluence.** Module `Path/Rewrite/Confluence.lean` builds `Confluence.Join` objects for every critical pair in the Knuth–Bendix table. Eliminating a critical pair yields a common reduct plus proofs that each branch rewrites to it. These joins feed a standard Newman-style proof of global confluence.

2.4 An illustrative rewrite

Consider a context $C[-]$ and a path $p : \mathsf{Path}\, A\, a\, b$. Rule (tsbll) of [**?**] states that

$$\mathsf{trans}\Big(p, \mathsf{subL}(\mathsf{refl}, s)\Big) \;\longrightarrow\; \mathsf{subL}(p, s). \tag{1}$$

In Lean the term

```
Step.context_subst_left_beta
  (C := context)
  (h := p) (k := s)
```

realises (**??**). If subL is itself defined via transports, equation (**??**) manifests as a dependent transport reduction. This explicit witness is essential when we later lift rewrites to quotients and build loop groups.

3 Lean 4 Formalisation Architecture

3.1 Project structure

The repository follows the hierarchy sketched in Figure **??**. Core path constructions live in `ComputationalPaths/Path/Basic`. Rewrite layers reside under `Path/Rewrite`, and homotopy-facing modules live under `Path/Homotopy` and `Path/HIT`. A dedicated documentation folder records the mapping between LNDEQ rules and Lean constructors, ensuring traceability between the SAJL article and the mechanised calculus.

3.2 From paths to quotients

The type $\mathsf{Path}\, A\, a\, b$ is defined inductively with constructors paralleling the rules of equality: reflexivity, symmetry, transitivity, congruence for unary and binary contexts, and substitution through dependent types. Its quotient by RwEq is denoted Path/RwEq and called `PathRwQuot`. The quotient exposes groupoid operations directly:

Module family	Purpose
`Path/Basic`	Reflexivity, symmetry, transitivity, substitution, transport, and congruence for computational paths.
`Path/Rewrite`	‘Step‘, ‘Rw‘, ‘RwEq‘, instantiations, termination, confluence, and automation tactics.
`Path/Groupoid`	Weak category, groupoid, bicategory, and weak two-groupoid interfaces built from paths and rewrites.
`Path/Homotopy`	Loop spaces, rewrite quotients, loop monoid/group instances.
`Path/HIT`	Circle and torus higher-inductive interfaces, encode/decode maps, and step lemmas.

Figure 1 – High-level organisation of the Lean development.

- Composition is induced by ‘cmpA‘, respecting associativity witnesses coming from rewrite constructors such as (tt) and (tsblr).
- Inverses are provided by symmetry and the ‘invA‘ constructor, with the rewrite rule (ss) certifying involutivity.
- Unit laws hold by construction because ‘refl‘ classes serve as identity morphisms.

Lean’s quotient API ensures that all proofs about Path/RwEq reduce to rewrite witnesses on raw paths, preventing the need for proof-irrelevant assumptions.

3.3 Automation

The ‘Instantiation‘ mechanism records the reason for each rewrite step. Together with the ‘Rw‘ closure, this enables reflective tactics: we can reify a goal of the form $\mathsf{RwEq}\,p\,q$, search for a derivation, and hand the resulting ‘Rw‘ back to Lean’s kernel. Two tactics stand out:

`rwEq_auto`
: Orchestrates transitivity, symmetry, context congruence, and substitution steps to prove RwEq goals automatically.

`twoCell_auto`
: Works at the bicategory level, combining ‘rwEq$_a$*uto*‘*withwhiskeringlemmastodischargecoheren*

These tactics dramatically reduce the overhead of managing rewrite evidence when proving encode–decode lemmas or transport laws. They also enable a declarative proof style that mirrors the paper proofs found in [**?**, **?**].

3.4 Integration with Lean's metaprogramming

Lean 4's macro and elaboration framework [**?**] allows us to encapsulate rewrite reasons inside attributes. For instance, each LNDEQ rule is registered as a simp lemma for the 'RwEq' simplifier, so that the standard 'simp' tactic can already solve many rewrite goals. Custom elaborators instantiate contexts automatically, ensuring that proofs resemble the high-level description of the rule instead of the low-level paths.

3.5 Engineering statistics

Running a clean build via `lake.cmd build` produces all modules in under a minute on a desktop-class machine, and the repository currently contains 12,817 lines of Lean spread across 29 files. The rewrite modules account for roughly one third of those lines, with homotopy-facing code occupying another third. The remainder includes support lemmas, groupoid packaging, and automation.

4 Homotopical Applications

4.1 Weak bicategories and two-groupoids

By packaging computational paths into categorical structures we obtain a weak bicategory `pathsBicat` and a weak two-groupoid `pathsTwoGroupoid`. Objects are the elements of A, 1-cells are path quotients, and 2-cells are rewrite classes. Whiskering, associators, unitors, and the interchange law all follow from rewrite constructors registered as coherence data. Consequently, the bicategory laws do not require postulated axioms: every coherence witness is a named Lean term derived from the LNDEQ rulebook.

4.2 Loop spaces and groups

Given a pointed type (A, a_0) we define the loop space $\mathsf{Loop}(A, a_0) := \mathsf{Path}(a_0, a_0)/\mathsf{RwEq}$ and equip it with a multiplication inherited from path composition, inverses induced by symmetry, and a neutral element given by refl_{a_0}. The resulting structure satisfies the group axioms strictly because associativity, unit, and inverse laws are witnessed by rewrite constructors such as (tt), (rrr), and (ss). The Lean record `LoopGroup` packages this data and serves as the entry point for encode–decode arguments.

4.3 Higher inductive interfaces

For the circle and the torus we postulate higher-inductive interfaces consisting of constructors, eliminators, and computation rules. Although Lean 4 does not yet offer native higher-inductive types, we axiomatise just enough to state the encode–decode maps and

to transport loops across the constructor equations. The computational-path machinery then carries the homotopical weight: all group laws, loop concatenations, and transport properties are proved via rewrites instead of pattern-matching on HIT constructors.

4.4 Encode–decode for $\pi_1(\mathbb{S}^1)$

The circle module defines:

- A base point `base` and a fundamental loop `loop`.
- A code family `circleCode : S1 → Type` that sends `base` to $\mathbb{Z}$.
- Maps `circleEncode : LoopGroup S1 base → Z` and `circleDecode : Z→ LoopGroup S1 base`.

Transport along the loop adds or subtracts 1 on the integer code, and the encode–decode equations

$$\texttt{circleEncode}\circ\texttt{circleDecode} = \mathsf{id}_{\mathbb{Z}} \quad \text{and} \quad \texttt{circleDecode}\circ\texttt{circleEncode} = \mathsf{id}_{\mathsf{Loop}(S1,base)}$$

hold up to canonical rewrites. Therefore $\pi_1(\mathbb{S}^1)$ is isomorphic to $\mathbb{Z}$ inside Lean. The proof follows the HoTT blueprint [**?**] but replaces equality induction with the LNDEQ rewrite tactics, which provide better automation when commuting transports and concatenations.

4.5 Encode–decode for $\pi_1(\mathbb{T}^2)$

The torus module defines two commuting loops and a code family targeting $\mathbb{Z} \times \mathbb{Z}$. The loop group maps encode traversals of the meridian and longitude, while the decode map rebuilds the corresponding rewrite class. Thanks to the context-sensitive rewrite rules (notably (ttsv) and (tstu)) we can prove that the loops commute strictly in the quotient, which is essential to show that the fundamental group is free abelian of rank two. The equivalence

$$\pi_1(\mathbb{T}^2) \cong \mathbb{Z} \times \mathbb{Z}$$

then follows from the same encode–decode strategy.

5 Labelled Natural Deduction and Fundamental Groups

The labelled natural deduction calculus introduced in [**?**] uses computational paths to derive Seifert–van Kampen style presentations of fundamental groups. Each homotopy class of loops corresponds to a rewrite name in the deduction system, and the Seifert–van Kampen theorem translates to a normalisation argument in the rewrite calculus. We import this perspective into Lean in three ways.

5.1 Reconstructing the Klein bottle

The Klein bottle $\mathbb{K}^2$ admits a presentation $\langle a, b \mid a^{-1}ba = b^{-1}\rangle$. In LNDEQ the key rewrites encode the gluing of edges with opposite orientations. By instantiating the Lean rewrite constructors with the same labels as in [**?**], we obtain a loop group whose normal forms match the classical presentation. The automation described earlier can simplify arbitrary composites of a and b to a canonical form, mimicking the calculations carried out on paper.

5.2 Connected sums and higher-genus tori

Chapter 4 of the SAJL paper computes the fundamental group of the two-holed torus $\mathbb{M}_2 = \mathbb{T}^2\#\mathbb{T}^2$ by assembling rewrite rules for each cell complex. Translating this construction to Lean requires two ingredients: (1) contexts that inject rewrites into connected components, and (2) transports showing how identifications propagate through dependent eliminators. Both pieces already exist in the Lean development thanks to the dependent context machinery, so the paper's derivations become short Lean proofs. Although we have not yet formalised the higher-genus surfaces axiomatically, the rewrite toolkit is expressive enough to follow the same reasoning.

5.3 From labelled proofs to Lean scripts

To reduce the gap between labelled deductions and Lean code we provide a small DSL whose syntax matches the rule mnemonics (e.g. `tt`, `tsbll`, `mx2l1`). A labelled proof in the sense of LNDEQ can thus be pasted verbatim into Lean, and the interpreter turns it into an instantiation record plus a 'Rw' derivation. This bridge is crucial for reusing the calculations reported in [**?**] without rewriting everything in tactic form.

6 Evaluation

6.1 Methodology

We assess the development along three axes: (1) coverage of the LNDEQ rewrite rules, (2) ability to derive homotopical theorems, and (3) ease of automation. The coverage is documented in `docs/LNDEQ.md`, which matches every rule against its Lean constructor. Homotopical theorems are represented by the encode–decode results discussed in Section **??**. Automation is measured by counting the number of rewrites discharged automatically in those proofs.

6.2 Quantitative data

- **Code size.** 12,817 lines across 29 Lean files (measured on November 22, 2025).
- **Rewrite coverage.** All forty base rules plus the two context-sensitive additions (ttsv) and (tstu) are implemented with named constructors, along with additional transports for dependent Σ-types.
- **Automation.** Over 80% of the rewrite steps appearing in the encode–decode proofs are produced by `rwEq_auto`; only the outermost structural arguments (e.g. induction on integers) require explicit scripting.

6.3 Qualitative observations

Two lessons emerged while porting the LNDEQ reasoning to Lean. First, the explicit instantiation of rules pays off: attaching names to derivations makes it easier to trace back to the SAJL article and to justify each step to the proof assistant. Second, quotient-based loop groups integrate smoothly with Lean's algebraic hierarchy once rewrite witnesses are available. This indicates that explicit computational paths can serve as a practical alternative to native HITs even before cubical features reach Lean.

7 Related Work

Our work builds on the foundational analyses of identity types by Martin-Löf [**?**] and by Hofmann–Streicher [**?**], and on the homotopical reinterpretation of type theory by Awodey, Voevodsky, and the HoTT community [**?**, **?**]. Compared to cubical or simplicial approaches, computational paths keep the syntax of equality intact while enriching the semantics with rewrite identifiers. The labelled natural deduction techniques of [**?**] adapt classical fundamental group computations, whereas our Lean development serves as a formal companion that validates every rewrite. Related mechanisations in Agda and Coq often appeal to higher inductive types directly; here we deliberately work at the level of rewrites to emphasise traceability with the original SAJL calculus.

8 Future Directions

The infrastructure described above opens several lines of research.

1. **Higher-inductive signatures.** We plan to extend the rewrite toolkit with generic schemas for point, path, and higher-path constructors so that new HITs can be added without reworking the automation.

2. **Cubical back-end.** Lean's forthcoming cubical kernel could supply judgmental computation rules for higher constructors. Integrating computational paths with cubical transport would combine the best of both worlds.

3. **Expanded labelled libraries.** The DSL that interprets LNDEQ derivations can be enriched with tactic-style macros, enabling proof scripts that stay close to the SAJL prose.

4. **Applications to higher-genus manifolds.** Ongoing work aims to formalise the n-holed torus and other connected sums by reusing the context-sensitive rewrites introduced here.

9 Conclusion

We have presented a Lean 4 formalisation of computational paths that faithfully reflects the LNDEQ rewrite system and supports homotopical reasoning. By bridging labelled natural deduction and proof assistant technology we obtain a reusable kernel for transport, rewrite, and quotient reasoning, capable of rederiving classical computations of fundamental groups. The development demonstrates that explicit rewrite identifiers are not merely a conceptual aid: they can be encoded, automated, and composed at scale, providing a viable foundation for future extensions of homotopy type theory inside Lean.