The Lax Whitehead Theorem for Coglobular Locally Compact Locales

```
/-
-/
```

```
/-
...
-/
```

Shanghe Chen and Dean Young

We wish to acknowledge the collaborative efforts of Shanghe Chen and Dean Young. Together the authors are pursuing these plans as a long term project.

1. Contents

2. Introduction

3. Unicode

Lean 4 uses unicode, and this entails an extensive catalogue of characters to choose from. Here is a list of the unicode characters we will use:

Symbol	Unicode	VSCode shortcut	Use
Lean's Kernel			
×	2A2F	\times	Product of types
→	2192	\rightarrow	Hom of types
⟨,⟩	27E8,27E9	\langle,\rangle	Product term introduction
↦	21A6	\mapsto	Hom term introduction
∧	2227	\wedge	Conjunction
∨	2228	\vee	Disjunction
∀	2200	\forall	Universal quantification
∃	2203	\exists	Existential quantification
¬	00AC	\neg	Negation
Variables and Constants			
	1D52,1D56		Variables and constants
$^0,^1,^2,^3,^4,^5,^6,^7,^8,^9$	1D52,1D56		Variables and constants
$^-$	207B		Variables and constants
$_0,_1,_2,_3,_4,_5,_6,_7,_8,_9$	2080 - 2089	\0-\9	Variables and constants
α-ω,A-Ω	03B1-03C9		Variables and constants
Categories			
𝟙	1D7D9	\b1	The identity morphism
∘	2218	\circ	Composition
Twocategories			
1	1D7CF		Horizontal identity map
•	2022	\smul	Horizontal composition of objects
∙	2219		Horizontal composition of morphisms
Adjunctions			
⇄	21C4	\rightleftarrows	Adjunctions
⇆	21C6	\leftrightarrows	Adjunctions
.	1BC94		Right adjoints
˙	0971		Left adjoints
⊣	22A3	\dashv	The condition that two Functors are adjoint
Monads and Comonads			
?,¿	003F, 00BF	?,\?	The corresponding (co)monad of an adjunction
!,¡	0021, 00A1	!, \!	The (co)-Eilenberg-(co)-Moore adjunction
ꜝ,ꜞ	A71D, A71E		The (co)exponential map
Miscellaneous			
∼	223C	\sim	Homotopies
≃	2243	\equiv	Equivalences
≅	2245	\cong	Isomorphisms
∞	221E	\infty	Infinity categories and infinity groupoids
∞	221E	\infty	Infinity categories and infinity groupoids

Of these, the characters ꜝ, ꜞ, ., ˙, **1**, and • do not have VSCode shortcuts, and so we provide alternatives for them.

It is not possible to copy the from the pdf to the clipboard while preserving the integrity of the code. To see the official Lean 4 file please click the link on the top right of the front page or click this link.

CATEGORIES AND HILBERT SPACES

4. Chapter 1: Resources for learning Lean 4

Before we get started defining what a Category is, we will cover the basic features of types in Lean 4. The main way to tell Lean 4 what something means is with `def`, which defines a term in dependent type theory. Much in the same way as other computer languages, we then supply the type of the term (e.g. `Int` for integer), followed by the formula itself:

Lean 1

```
def n : Int := 1
```

Here we have introduced an integer `n` using the type `Int` that comes with Lean 4. In addition to defining terms, Lean allows you to perform computations and prove theorems. It provides a rich set of tools for reasoning about mathematical statements. As a beginner, it's normal to take some time to get comfortable with Lean and formal proof systems. It's a journey that requires practice and patience. Lean has an active community that provides support and resources to help you along the way.

We encourage you to explore Lean further and experiment with different definitions and proofs. Start with simple examples and gradually build your understanding. The Lean documentation and online tutorials can be valuable resources to learn more and deepen your knowledge.

Constituents of `x, y : X` of types `X` can also stand to be equal or unequal, written `x = y`, and it is the properties of equality which in addition to the dependent type theory make a type behave like a set. Equality satisfies the three properties of an equivalence relation, which we cover presently. Consider first the reflexivity property of equality:

Lean 2

```
def reflexivity {X : Type} {x : X} (p : x = x) :=
↪  Eq.refl p
```

This command defines a function called reflexivity that proves the reflexivity property of equality. The function takes two type parameters: `X` represents the type of the elements being compared, and `x` represents an element of type `X`. It also takes an argument `p` which is a proof that `x` is equal to itself (`x = x`). The function body states that the result of `reflexivity` is the proof `p` itself using the `Eq.refl` constructor, which indicates that `x` is equal to itself.

In `Lean 4`, `{x : X}` represents an implicit argument, where Lean will attempt to infer the value of `x` based on the context. `(x : X)` represents an explicit argument, requiring the value of `x` to be provided explicitly when using the function or definition.

Lean 3

```
def symmetry {X : Type} {x : X} {y : X}  (p : x = y)
 ↪  := Eq.symm p
```

This command defines a function called symmetry that proves the symmetry property of equality. It takes three type parameters: `X` represents the type of the elements being compared, and `x` and `y` represent elements of type `X`. The function also takes an argument `p` which is a proof that `x` is equal to `y` (`x=y`). The function body states that the result of `symmetry` is the proof `p` itself using the `Eq.symm` constructor, which allows you to reverse an equality proof.

Lean 4

```
def transitivity {X : Type} {x : X} {y : X} {z : X}
 ↪  (p : x = y) (q : y = z) := Eq.trans p q
```

This command defines a function called transitivity that proves the transitivity property of equality. It takes four type parameters: `X` represents the type of the elements being compared, and `x`, `y`, and `z` represent elements of type `X`. The function also takes two arguments `p` and `q`. `p` is a proof that `x` is equal to `y` (`x = y`), and `q` is a proof that `y` is equal to `z` (`y = z`). The function body states that the result of transitivity is the proof of the composition of `p` and `q` using the `Eq.trans` constructor, which allows you to combine two equality proofs to obtain a new one.

These `Lean` commands define functions that prove fundamental properties of equality: reflexivity (every element is equal to itself), symmetry (equality is symmetric), and transitivity (equality is transitive). These properties are essential for reasoning about equality in mathematics and formal proofs.

We must also require that functions satisfy extensionality:

Lean 5

```
def extensionality (f g : X → Y) (p : (x:X) → f x =
↪  g x) : f = g := funext p
```

Extensionality, a key characteristic of sets and types, asserts that functions which are equal on all values are themselves equal, and it is featured prominently in what is perhaps the most well known mathematical foundations of ZFC.

There are several other features of equality with respect to functions which we should be aware of:

Lean 6

```
def equal_arguments {X : Type} {Y : Type} {a : X} {b
↪  : X} (f : X → Y) (p : a = b) : f a = f b :=
↪  congrArg f p

def equal_functions {X : Type} {Y : Type} {f₁ : X →
↪  Y} {f₂ : X → Y} (p : f₁ = f₂) (x : X) : f₁ x =
↪  f₂ x := congrFun p x

def pairwise {A : Type} {B : Type} (a₁ : A) (a₂ : A)
↪  (b₁ : B) (b₂ : B) (p : a₁ = a₂) (q : b₁ = b₂) :
↪  (a₁,b₁)=(a₂,b₂) := (congr ((congrArg Prod.mk) p)
↪  q)
```

The tutorial here provides a good introduction to using the dependent type theory in Lean. Evgenia Karunus has created an exposition of the Curry-Howard correspondence in Lean 4 here, as well as a list of tutorials here.

5. Chapter 2: Categories

Section	Description
Category	the Category structure
Cat	the Category of categories
Set	the Category of sets

Bibliography

1. Arlin, Kevin David. “2-categorical Brown representability and the relation between derivators and infinity-categories.” Doctoral dissertation, University of California, Los Angeles, 2020.

2. Saunders Mac Lane, "Categories for the Working Mathematician," Graduate Texts in Mathematics, vol. 5, Springer-Verlag, New York, 1971.

3. Samuel Eilenberg and Saunders Mac Lane, "General Theory of Natural Equivalences," Transactions of the American Mathematical Society, vol. 58, no. 2, pp. 231-294, 1945.

4. Daniel M. Kan, "Adjoint Functors," Transactions of the American Mathematical Society, vol. 87, no. 2, pp. 294-329, 1958.

5. Chris Heunen, Jamie Vicary, and Stefan Wolf, "Categories for Quantum Theory: An Introduction," Oxford Graduate Texts, Oxford University Press, Oxford, 2018.

6. S. Eilenberg and J. C. Moore, "Adjoint Functors and Triples," Proceedings of the Conference on Categorical Algebra, La Jolla, California, 1965, pp. 89-106.

7. Daniel M. Kan, "On Adjoints to Functors" (1958): In this paper, Kan further explored the theory of adjoint Functors, focusing on the existence and uniqueness of adjoints. His work provided important insights into the fundamental aspects of adjoint Functors and their role in Category theory.

8. Leonardo de Moura and Jeremy Avigad, "The Lean Theorem Prover," Journal of Formalized Reasoning, vol. 8, no. 1, pp. 1-37, 2015.

9. Riehl, Emily, and Dominic Verity. Elements of ∞-Category Theory. Johns Hopkins University, Baltimore, MD, USA and Centre of Australian Category Theory, Macquarie University, NSW, Australia.

Further reading:

1. J. Beck, "Distributive laws," in Seminar on Triples and Categorical Homology Theory, Springer-Verlag, 1969, pp. 119-140.

2. Leonardo de Moura and Soonho Kong, "Lean Theorem Proving Tutorial," Proceedings of the 6th International Conference on Interactive Theorem Proving (ITP), Lecture Notes in Computer Science, vol. 9236, pp. 378-395, Springer, Berlin, 2015.

3. Jeremy Avigad, Leonardo de Moura, and Soonho Kong, "Theorem Proving in Lean," Logical Methods in Computer Science, vol. 12, no. 4, pp. 1-43, 2016.

4. Daniel Selsam, Leonardo de Moura, David L. Dill, and David L. Vlah, "Leonardo: A Solver for MIP and Mixed Integer Nonlinear Programming," Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS), pp. 493-504, 2019.