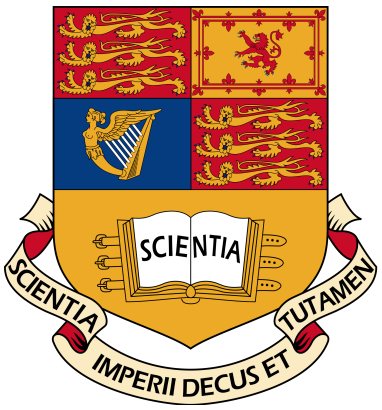
Schemes in Lean

CO401J Project - Final Report

19th of June 2019

Imperial College London



Author
Ramon Fernández Mir

Supervisor
Prof. Kevin Buzzard

Second Marker
Dr. Zaniar Ghadernezhad

Abstract

In this report we present a formal de nition of Grothendieck's notion of a scheme [1, 01IJ] using the Lean Theorem Prover and explain rigorously all the steps of the process.

Acknowledgements

I would like to thank my supervisor, Kevin Buzzard, for giving me the opportunity to contribute to this new era of mathematics and teaching me how to think like a mathematician. His passion for mathematics and his determination to bring formal mathematics to the next level have been key to the success of this project.

I am also thankful to the Lean community. Whenever I got stuck I knew I could simply ask them and my question would rapidly be answered by very bright people who showed me a genuine interest to move this project forward. In particular, I would like to mention two undergraduate students here at Imperial, Kenny Lau and Chris Hughes, who not only are incredibly talented and major contributors to the formalisation of mathematics in Lean but also hugely helped setting up the foundations of this project.

Throughout this project, there have been a few good friends that have spent more hours than they thought they ever would hearing about schemes and Lean. I thank Mart , Aris and Cristobal for their patience and for these wonderful four years.

Finally, I would like to thank my family, especially my parents Mª Carmen and Juan Carlos, and my siblings Elia and Joan for their love and support throughout my degree.

Contents

Chapter 1

Introduction

1.1 Formalising mathematics

Mathematical proofs are normally written in natural language under the assumption that, given enough time, every step could be spelled out as a logical formula. Then one could check that each step is deduced from the previous ones following a simple set of rules. Computers are particularly good at checking straightforward logical implications and, thanks to them, it is no longer just an assumption but a reality.

The earliest work on computer-assisted theorem proving took place around the 1950s [2]. Back then, the focus was on completely automated theorem proving, by which we mean that there is no user input while the theorem is being proved. There was a lot of interest in the field of artificial intelligence, which had just been born, and, moreover, computers had very primitive interactive features. For these reasons, it took some years until a more interactive approach was taken. Introducing interaction was a big step forwards towards the aim of formalising mathematics. The idea is to have a programming language to write proofs in a style that resembles natural language and use the system as a proof assistant that checks the steps. Automation still plays an important role and combining both philosophies is one of the keys to the success of this field of research.

In 1967, De Brujin started the Automath project, which is a remarkable milestone [3]. He used the idea that proofs can be seen as objects and formulas as types. To prove that a formula is valid one needs to give a proof object of such type. The proof assistant acts then as a type checker. This correspondence is known as the Curry-Howard isomorphism. Apart from some exceptions such as Mizar or Isabelle/ZF, which are based on set theory, most of the mainstream proof assistants such as Isabelle/HOL, HOL Light, Coq or Lean rely heavily on type theory. The rest of this section is devoted to going through some of the biggest achievements using these systems.

An interesting motivating story is that of the proof of the Kepler's conjecture. Phrased simply, it says that the most efficient way to pack oranges is to arrange them in hexagons. Thomas Hales claimed to have proved it in 1998 [4]. It took a team of twelve experts about four years to go through the proof only to conclude that they were pretty sure that it was correct. A couple of years later, in 2005, an even more detailed proof was given and their concerns were dropped so the proof was finally published [5]. At that point Hales had already started

the Flyspeck project, a collaborative project whose goal was to formally verify his proof. In August 2014 he announced that the proof had been successfully veri ed using a combination of Isabelle and HOL Light and it was accepted in 2017 [6]. This story illustrates how formalising mathematics can play a role in the veri cation of controversial proofs. Another takeaway, however, is that it is a highly nontrivial task and that there is a long road ahead of us if the purpose is for the standard mathematician to adopt these technologies.

Both HOL Light and Isabelle/HOL are based on higher-order logic. The former was written and is maintained by John Harrison. It stands out for having a remarkably small kernel (about 400 lines of code) [7]. The latter is the most common extension of Isabelle's logical framework. Isabelle was created by Lawrence Paulson and a vast amount of mathematics has been formalised using it. As of June 2019 there are 475 articles in Isabelle's Archive of Formal Proofs. As an example, a complete proof of Gödel's incompleteness theorems is given in [8].

Any historical review of formalised mathematics would be incomplete without mentioning Georges Gonthier. He led the formalisation of two big theorems using the Coq proof assistant. In 2008 he published a formal proof of the four-colour theorem, which says that any map (in the cartographic sense) can be coloured using four colours [9]. Five year later, he and his team nalised the formal proof of the odd order theorem [10]. This theorem says that every nite group of odd order is solvable and is a major step forward towards the formalisation of the classi cation of nite simple groups. Coq's foundation is known as the Calculus of Inductive Constructions, based on dependent type theory. These are important concepts that will be introduced properly and discussed in section 2.1.

The last theorem prover to join the family has been Lean. The project was launched in 2013 by Leonardo de Moura at Microsoft Research. Lean appears after a thorough analysis of the advantages and disadvantages of the proof assistants mentioned above. It aims at using the many advantages of dependent type theory without jeopardizing automation. In a short period of time, it has raised a lot of interest in the formal mathematics community and a great example of that is mathlib. It is maintained by Jeremy Avigad, Reid Barton, Mario Carneiro, Johan Commelin, Sebastien Gouëzel, Simon Hudon, Chris Hughes, Robert Y. Lewis and Patrick Massot, and it has grown exponentially since its creation in 2017 (see section 2.3 for more details). Some papers on theorems formally veri ed in Lean have started to appear such as the one written by Robert Y. Lewis on the p-adic numbers and Hensel's lemma [11] or the formalisation of the cap set problem [12] written by the author above together with Sander R. Dahmen and Johannes Hölzl. Finally, it is worth pointing out that Lean is the language chosen in the Formal Abstracts Project (`https://formalabstracts.github.io/`), a project led by Hales with the mission of creating service to give formal statements of the main theorems presented in papers in the philosophy of the QED manifesto [13].

1.2 What is a scheme?

The word *scheme* is used for the rst time in the mid-1950s by Chevalley and Cartier in a much more restricted sense. Their discussions inspired Grothendieck to start an incredibly ambitious program whose goal was to establish general enough foundations for algebraic geometry [14]. His work is gathered in several fascicles which compose *Elements de geometrie*

algebrique, published between 1960 and 1967. The de nition of a *scheme* (or rather, in his notation, a *prescheme*) is de nition 2.1.1 in Chapter 2 of the rst volume [15]. A pure mathematics student is typically introduced to the notion of a scheme in a Master's level algebraic geometry course. We will try to provide a, for now, super cial understanding of the origin and importance of this de nition.

A ring is, roughly speaking, a structure where we can perform addition and multiplication. We may assume that multiplication is commutative. Given a topological space X, one can consider all the continuous functions $f : X \to \mathbb{R}$ and these form a commutative ring. An algebraic study of this ring can be used to recognise geometric properties of X. With this idea in mind, Grothendieck de ned a geometric object $Spec(R)$ such that its ring of `functions' was isomorphic in a natural, yet not obvious, way to the ring R. This is the idea behind an *a ne scheme* and their importance relies on the fact that they can be put together to form a category which, in a very speci c sense, corresponds to the category of commutative rings. Therefore, we can view the elements of a ring as `functions', which might be intuitive in rings such as $\mathbb{C}[x;y]$ but not so much in, for instance, $\mathbb{Z}$. A nice way to think about it is to see a ne schemes are the geometric objects where we can do algebra, by which we mainly mean solving equations [16]. A concept that appears frequently in many areas of mathematics is that of locality. A manifold, for example, is a space where you can locally do calculus. They were de ned in the 19^{th} century and are a great example of the power of this notion in analysis. We would like to extend this principle to algebra. A *scheme* is a topological space that can be covered by a ne schemes, i.e. a space where we can locally do algebra.

Modern algebraic geometry is the study of schemes. Before Grothendieck, most of the theory was built on algebraic varieties. Suppose we are working over $\mathbb{C}$ and $p(x) \in \mathbb{C}[x]$ then the points $a \in \mathbb{C}$ where $p(a) = 0$ form a variety. In the language of schemes, one would instead look at the prime ideals of $\mathbb{C}[x]/(p(x))$. This simple change of approach already presents many advantages. The polynomials $p_1(x) = x$ and $p_2(x) = x^2$ de ne the same variety, namely $\{0\}$. However, they de ne di erent schemes, $\{(0)\}$ and $\{(x)\}$ respectively. Schemes are hence more general than varieties and working in this framework is sometimes indispensable to solve certain problems. A good example of this phenomenon are the Weil conjectures which were one of the main motivations for Grothendieck to develop this new formalism. There are many other notable theorems where scheme theory plays a fundamental role, especially in Number Theory. As a matter of fact, Wiles' proof of Fermat's Last Theorem uses scheme theory in a crucial way. Of course, understanding how is way beyond the scope of this report. As we will see, we will struggle enough to simply de ne a scheme and make all the intuition given above precise.

1.3 Objectives

The starting point of this project is the work done by Kevin Buzzard, Chris Hughes and Kenny Lau on formally verifying parts of The Stacks Project [1]. In particular, they formalised enough to give a de nition of an object mathematically equivalent to a scheme. Along the way, they realised that a lot of the machinery they needed had not been written in Lean so they lled those gaps and made them available to the community by putting them in mathlib, the Lean mathematical library, whose contents are summarised in section 2.3. They also encountered many technical issues which, in some cases, were solved by brute force and

hence led to a very large codebase (almost 10000 lines of code) with fairly unmanageable parts. The code can be found in `https://github.com/kbuzzard/lean-stacks-project`.

Having learnt from the mistakes made in that first attempt and with a better idea of how to solve them, the objective of this project is to, starting from scratch, define a scheme in Lean. This time the definition should be the one given in [1, 01IJ] in the same level of abstraction without skipping any definition in the process. Moreover, the codebase should be cleaner and more organised, finding better ways to tackle the numerous technical difficulties.

1.4 Contributions

This project gives the first complete formal definition of a scheme ever written in a theorem prover. Even though a large amount of mathematics, including incredibly complicated results, has been successfully formalised, we realised that often the focus was on the proofs rather than the object themselves. The underlying objects tend to be graphs, groups or other elementary objects. By formalising a highly complex object such as a scheme we have shown that the field of formal verification is mature enough to handle modern abstract mathematics.

Regarding the objectives stated in the previous section, they have been satisfactorily accomplished. We give a list of the ways in which our project has improved its predecessor:

> The definition of a scheme is not only mathematically correct but also the standard one, which we have achieved by introducing the concept of a locally ringed space. For us, a scheme is a locally ringed space that is covered by affine schemes, which are also locally ringed spaces (see section 3.5). Before, a scheme was defined as a topological space with a sheaf of rings satisfying a property equivalent to being covered by affine schemes.
>
> Our approach on localisation of rings has heavily simplified some of the main proofs. This is explained in detail in section 3.1.1. Neil Strickland proposed three axioms to characterise ring localisation and we use his predicate instead of the explicit construction throughout. Thanks to that, we avoid convoluted arguments about `canonically isomorphic' structures.
>
> The files are sensibly organised and the same naming convention is followed throughout. By doing so, we provide a usable interface for any Lean user that would need to use our definition of a scheme. For example, our work on sheaf theory, found in section 3.2.2, is currently being used in the Perfectoid Spaces Project in which Kevin Buzzard, Patrick Massot and Johan Commelin have given a formal definition of a perfectoid space, a concept introduced by field medalist Peter Scholze in 2012. Their definition is publicly available in `https://github.com/leanprover-community/lean-perfectoid-spaces/`.
>
> Two pull requests have been merged into mathlib:
>
> - We extended a basic result about group homomorphisms to additive groups (`https://github.com/leanprover-community/mathlib/pull/947`).
> - We further extended the same result to ring homomorphims (`https://github.com/leanprover-community/mathlib/pull/951`).

This project has been mentioned in three talks given by Buzzard:

His talk *Schemes in Lean* in the 4th conference on Artificial Intelligence and Theorem Proving which took place on April 11th 2019 (http://aitp-conference.org/2019/slides/KB.pdf) was based on the development of this project.

In his talk *Mathematical objects in dependent type theory* on May 29th 2019 at the Big Proof workshop (http://wwwf.imperial.ac.uk/~buzzard/docs/buzzard_big_proof2019.pdf) he used some of the issues encountered as examples.

In May 30th 2019 he gave a talk for the 80th anniversary of the CNRS held by the CNRS-Imperial Abraham de Moivre UMI (https://www.imperial.ac.uk/news/191573/cnrs-imperial-umi-workshop-public-lecture-showcases/). It was titled *The Future of Mathematics* and it briefly mentioned this project to show that it is possible to formalise complex mathematical objects.

Finally, we believe that formally defining a scheme is an important achievement especially since it had never been done before. We are currently in the middle of talks about writing a paper essentially summarising this report as we think that it is a valuable experience to share with the formal verification community.

1.5 Report layout

We diverge slightly from the standard layout. There is no `Evaluation' section as, due to the nature of this project, there are no meaningful performance metrics to be discussed. Also, there is no existing work that we can directly compare to except from the previous formalisation attempt. The main differences were already stated in section 1.4 and we will highlight them as we encounter them in Chapter 3. That being said, we are aware of the limitations which, in this case, fall under the `Future Work' chapter.

The structure of this report is as follows. In Chapter 2 we introduce Lean both as a theorem prover and as a programming language. We explain its underlying formalism, give an idea of how to use it and summarise the contents of mathlib relevant to us. Chapter 3 contains the mathematical development all the way to the definition of a scheme. Firstly we develop the theory of ring localisations explaining our own particular approach and proving the results needed. We also define $Spec(R)$ and prove its main properties. We proceed by covering the necessary sheaf theory including sheaves on bases and extensions. Finally, we define locally ringed spaces, prove that $Spec(R)$ has the structure of a locally ringed space and present our definition of a scheme. Chapter 4 proposes a plan to properly test our API and ensure its robustness. After that, we give some concluding remarks on the work done in Chapter 5.

Chapter 2

Theorem Proving in Lean

In the sections that follow we provide a concise introduction to Lean. Chapter 3 includes a considerable amount of code in Lean as well as references to Lean's components. Our aim is to provide enough background to be able to follow it. This chapter is based on the two main references for programming in Lean: *Theorem Proving in Lean* [17] and *The Lean reference manual* [18]. Other references where some ideas and examples in this chapter come from are [19] and Chapter 2 of [20].

2.1 Dependent type theory

One of the core ideas of Lean's underlying formalism is Martin-Löf's intuitionistic type theory. It is an alternative foundation of mathematics based on constructivism. In order to give a proof, one needs to provide a `witness' so, essentially, proofs by contradiction are not allowed. His initial formulation introduces *dependent types* which are, loosely speaking, types that take other types as inputs [21]. For example, suppose we wanted to define the type `list`. We would like to be able to write `list ℕ` for the type of lists of natural numbers and `list ℝ` for the type of lists of real numbers. Now, what would be the type of a function like `reverse` that takes a list of any type and reverses it? This can be expressed in the language of dependent type theory as the *pi type* `Π α : Type, list α → list α`.

From this idea, a whole family of dependent type theories was born. Lean is based in one of them, known as the *Calculus of Inductive Constructions*. It is the same formalism as Coq's and is the most expressive in the family of typed lambda calculi by combining polymorphism, dependent types and datatypes. Type theory is a very rich area with many interesting subtleties and we refer the reader to [22] for a formal approach. Instead, we will focus on the case of Lean.

In Lean's Calculus of Inductive Constructions, types are first-class citizens. That means that not only every term has a type but also every type has a type. There are fundamentally two ways to define types:

> Building a pi type. If `A` and `B` are types, we can define the type `Π x : A, B`, where `x` might appear freely in the type `B`. If it does not, we write `A → B`.

Inductively by specifying its constructors. The canonical example is the de nition of the type for natural numbers.

```
inductive nat : Type
| zero : nat
| succ : nat → nat
```

When a de nition such as the one above is given, Lean generates a *recursor* which, in this case, would be the principle of mathematical induction. There is an important class of inductive types, which are those with only one constructor and are given a special keyword: `structure`. Consider the following de nition.

```
inductive nat_pair : Type
| mk (fst : nat) (snd : nat) : nat_pair
```

It does not use recursion in any way which justi es why this notation is introduced.

```
structure nat_pair :=
(fst : nat) (snd : nat)
```

A notion that has been mentioned but not properly introduced is that of a term. We use the word *term* for any language construct, or *expression*, that can be typed. We split them into four groups: types, constants, lambda abstractions and applications.

Types. As we said, types are rst-class citizens. In the example above, for instance, `nat` has type `Type`. But we should also be able to give `Type` a type. This is achieved by providing a hierarchy of *universes*. More concretely, `Type` has type `Type 1`, `Type 1` has type `Type 2` and so on. By de nition, a type in Lean is an expression of type `Sort u` for some universe `u`. There are two special kinds:

The type of propositions `Prop := Sort 0`. This is the type where logical formulas live in. For example, call the following expression `FLT`.

```
∀ n : N, n ≥ 3 → ∃(∀ x y z : N, x^n = y^n + z^n ∧ x * y * z ≠ 0)
```

It has type `Prop`. If we manage to nd a term `p : FLT` we will have proved Fermat's Last Theorem in Lean! This exempli es the proposition-as-types correspondence. In this context, terms of a type `P : Prop` are called *proofs*. Lean does not distinguish between proofs of the same proposition, we say that `Prop` is *proof-irrelevant*. Another particularity is that any type de ned using types in `Prop` is of type `Prop`, known as *impredicativity*. As we will see shortly, this does not happen as we go higher in the hierarchy. The expression `P : Prop, P → P` has type `Prop` because we, and Lean, think of it as `∀ P : Prop, P → P`, a data-less (second-order) logical formula i.e. a proposition.

The type `Type := Sort 1` and, in general, `Type u := Sort (u+1)`. If `t` is of a type `T : Type u` we think of it as *data*. In comparison with `Prop`, terms behave quite di erently. Both `3` and `5` have type `nat` but they are certainly not equal in the eyes of

Lean. Moreover, universe levels might change. For example, the type of `T : Type u, T → T` has to be `Type (u+1)` to avoid paradoxes.

Constants. For any type `T`, we can add a constant `c` of type `T` into the language by writing `constant c : T`. Constants are sometimes added behind the scenes. For instance, when we defined `nat`, `nat.zero` automatically became a constant of type `nat`.

Lambda abstractions. Suppose `A` and `B` are types. If `b : B` is a term, possibly with a free variable `x` of type `A`, then `λ x : A, b` is a term. Its type is `Π x : A, B`.

Applications. Instead, assume `f : Π x : A, B` and `a : A`. Then we can combine them to obtain the term `f a`, of type `B`.

As a final remark, note that it is inferred from our discussion about `Prop` that any combination of terms inside `Prop` will remain in `Prop` so, in general, there is no way to generate data from propositions. This issue disappears if we assume the well-known axiom of choice which, in Lean, looks as follows:

```
axiom choice {α : Sort u} : nonempty (Π α)
```

We will assume the axiom of choice and work non-constructively. In particular, in this context, choice will allow us to get data from a proof. Suppose `p : A → Prop` and we have a proof `h` of type `∃ x : A, p x`, then we can write `classical.some h`, which is a term of type `A` satisfying `p` as asserted by `h`. This is Lean's way to incorporate non-constructive arguments to Martin-Löf's inuitionistic type theory. Note that, under this assumption, it does not make sense to think of proofs as algorithms as one could do in a constructive framework. In fact, whenever choice is used, the definition in question has to be marked as `noncomputable` because, since we have made data appear magically, it has no computational interpretation.

2.2 Language overview

So far we have seen the types and terms in Lean but have not discussed the variety of ways to input them as a front-end user. There are essentially two methods to do it: declaratively (writing terms explicitly) or interactively (using a sequence of *tactics*). In both cases, there are many ways in which Lean helps so that not every detail needs to be spelled out. The component in Lean's system whose task is to infer pieces of information not directly given by the user is called the *elaborator*. It is built on top of Lean's small trusted *kernel*, which is written in C++ and contains the type-checker as well as an API for manipulating terms and types.

In this section we discuss the declarative language and its subtleties, and explain what tactics are and how they are used in practice.

Declarations. We have already seen a few examples of the keywords used to assign a type to an identifier: `inductive` and its specialisation `structure`, and `constant` and its equivalent `axiom`. We would also like to be able to declare terms. This is done using the `def` keyword.

```
def one : nat := nat.succ nat.zero
```

In general, for any type T we can write def t : T = a where a is a term of type T. The alternative keywords lemma and theorem have the same e ect but are meant to be used when T is of type Prop.

As a nal remark about declarations, there is special syntax to de ne terms of a structure type which we frequently use. Instead of {field1 := a, field2 := b, ...} we can write ⟨a, b, ...⟩. In addition, we can use projections to access the values of the elds. For example, consider the structure and with two elds left and right. Using the in x notation ^ for and, suppose that H : A ^ B where A and B are of type Prop. We can write and.left H to obtain the part of the proof H that shows A or H.left or, more concisely, H.1.

Arguments and variables. The declarations above might take variables as inputs. These can be written right after the identi er as a list (x : X) (y : Y) ... which can then be used in the declaration. For example, we could write the following.

```
lemma and_comm (A B : Prop) (H : A ^ B) : B ^ A := ⟨H.2, H.1⟩
```

Alternatively, these variables can appear outside the de nition using the variables keyword, which is particularly useful if they are to be reused by other de nitions.

```
variables (A B : Prop) (H : A ^ B)
```

The scope of these variables would normally be the le they are in but it can be re ned by de ning *sections* or *namespaces*. The convention that we will follow in Chapter 3 is that the scope of the variables declared is always the section they appear in.

In the example above, the variables are declared explicitly, which means that, if we wanted to apply the lemma and_comm, we would have to give it three arguments. It is also possible to de ne variables implicitly and hope that Lean can infer them automatically. To make A and B implicit arguments, we would write {A B : Prop} instead of (A B : Prop). Now the lemma and_comm just takes one argument, namely H. The implicit arguments will be marked as *metavariables* when the expression is parsed so that the elaborator knows that it has to nd them. In this case, this was a sensible idea because the elaborator can easily infer A and B from H. Even if the argument is explicit, we can write an underscore `_' if we want to introduce a metavariable and make Lean do the work for us which, of course, is not always possible. There is one third major way of handling arguments, which is used when the type of the argument is a *type class*.

Type classes. When examples get complicated, it might be di cult for the elaborator to nd the right value for a metavariable. A way to solve this issue is to mark a family of types as a type class and then provide *instances*, which are elements in the type class and serve as hints. As an example, we consider the implementation of commutative rings in Lean. A type A is a commutative ring if all the axioms can be proved after choosing some zero : A, one : A, add : A → A → A and mul : A → A → A. Suppose we list all the axioms in a structure and call it comm_ring. A lemma that talks about a commutative ring then could take arguments (A : Type) (H : comm_ring A). If we want to apply it to Z then we would

have to also pass a proof that Z is a commutative ring, which feels strange. A rst idea would be to write {H : comm_ring A}. However, following with our example, it would be, in general, unrealistic to expect Lean to automatically prove all the axioms for Z if comm_ring is just a structure. The right way to do it is to annotate comm_ring as a type class. Then we write in Lean the proof that Z is a commutative ring using the keyword instance.

```
instance int_comm_ring : comm_ring Z
```

The arguments of the lemma then become (A : Type) [comm_ring A]. This tells Lean to use type class inference to deduce that A is a commutative ring which, in particular, means that it will look through the list of instances and nd int_comm_ring when A is Z.

Type classes are used extensively in mathlib as they provide the infrastructure to build the algebraic hierarchy. We would like the class of commutative rings to be included in the class of rings and this last one to be included in the class of groups. This is indeed the case and, if a lemma takes arguments (A : Type) [group A], we can apply it to Z straight away.

The last language feature that we explore is concerned with how a front-end user can enter terms into the system.

Tactics. As we mentioned at the beginning of the chapter, a term can be written using *tactics*. Tactics are instructions that modify existing terms or generate new ones using information in the current *environment* (known de nitions, lemmas and constants) and *local context* (list of hypotheses). They relieve the user from the tedious task of writing long and complicated terms. Moreover, they make the proofs more readable. They play a crucial role in interactive theorem proving. When we enter Lean's interactive mode, we see the local context and the current goal and see how they change as we use di erent tactics. We list some of the most important ones to give an idea of the sort of interaction that Lean allows:

> intro. When the current goal is a pi type x : A, B (or a universally quanti ed expression *8* x : A, B), we can use intro a to add a term a : A to the the local context. The new goal becomes B[a/x] that is, B replacing all occurrences of x by a.
>
> existsi. If the goal is an existentially quanti ed proposition *9* x : T, P x then invoking existsi t where t : T will change the goal to P t.
>
> exact. Given a term of type A and a goal of type B, this tactic solves the goal if the elaborator is able to unify A and B.
>
> cases. If one of our hypotheses is an inductive type, applying cases on it will create a goal for each constructor and expose its arguments.
>
> refl. We apply this tactic when the goal is of the form a b where is a re exive relation. In particular, we use it to solve a = b and it only works when the terms are *de nitionally equal*, i.e. can be reduced to a common term using the various reduction strategies in Lean's back-end.
>
> apply. This tactic tries to match the conclusion of an expression to the current goal. For example, if h : A ! B and the current goal is B, calling apply h will result in the goal becoming A.

`simp`. This is a very complex tactic with many variants but it is worth mentioning it as it is used quite frequently. In its most basic form, `simp` will try to use relevant lemmas and hypotheses to simplify the goal.

`rw`. As above, the inner workings of the rewriter are quite subtle so we will not go into much detail. The idea is that we can use know equalities or if-and-only-if's to change the type in the goal.

There are many other tactics and useful variations to the ones mentioned. A complete list can be found in Chapter 6 of [18]. Tactics can be used inside terms using the keyword `by` followed by the name of the tactic. In order to use them interactively, we use them inside a `begin ... end` block as shown below.

```
variable (R : N → N → Prop)

lemma forall_ex_of_ex_forall : (∃ x, ∀ y, R x y) → (∀ y, ∃ x, R x y) :=
begin
  intros H b,         -- H : ∃ x, ∀ y, R x y; b : N ⊢ ∃ (x : N), R x b
  cases H with a Ha, -- b a : N; Ha : ∀ y, R a y ⊢ ∃ (x : N), R x b
  existsi a,          -- b a : N; Ha : ∀ y, R a y ⊢ R a b
  exact (Ha b),       -- goals accomplished
end
```

Written as comments on the right hand side of the tactics we see the state of the proof at each step. That is what a Lean user sees when writing a proof in tactic mode. The tactic `intros` is just an extension of `intro` that takes multiple arguments. Another observation is that in here we can destruct an existentially quanti ed proposition without using the `classical` library. That is because the goal is of type `Prop` so there is no data involved. We say that types in `Prop` *eliminate to* other types in `Prop`. If the goal was of type `Type` we would need to use the axiom of choice.

2.3 mathlib

In order to understand the de nitions and lemmas stated in this report, it is important to explain some of the main parts of mathlib. In it we can nd anything from the rational numbers to complex analysis and it is in active development. We give an overview of the theories that we have used:

Sets. The de nition of a set is in Lean by default but in `data/set` we nd multiple useful lemmas about the *2* and relations. It also includes the theory of nite sets.

Relations and quotients. This is another example of an extension of Lean's core library. Lean includes the notion of a setoid, which are sets together with an equivalence relation and are used to de ne quotient types. Interestingly enough, quotients are dealt with di erently compared to other types. They are initially de ned in the kernel, which allows for de nitional equality in some of the common use cases. This part of mathlib provides another layer of API including, for instance, versions of some lemmas that do not use type class inference.

Topology. We find basic definitions such as the type class `topological_space`. It includes definitions and lemmas for continuity, compactness, bases and so on. As a remark, we often use the type opens which is a type built from a set together with the proof that it is open in a topological space.

Ring theory. This is perhaps the most relevant theory for this project and definitely the most used in our development. Amongst others, definitions for ideals, prime ideals, ring homomorphisms and ring localisations are provided. Moreover, each of these objects is equipped with a carefully designed set of lemmas.

Chapter 3

Schemes

This chapter is the core of the report. It contains the main implementation details and culminates with our definition of a scheme. We mainly follow [1] but Chapter 2 of [23] has been used as a guide for the steps to construct a scheme. The idea is to walk the reader through the codebase in a reasonable order. We combine mathematical definitions and lemmas in the usual style with their counterparts in Lean. In particular, Lean code for any statement that is new and not found in mathlib is provided. There are several reasons to do this. Even though this approach unavoidably introduces some repetition, it serves as a way to convince the reader that the definitions are correct. Moreover, it brings up interesting discussions about the design choices made and illustrates the similarities and differences between mathematicians and computers when defining mathematical objects. No proof in Lean is included but formal proofs of all the statements that appear can be found in `https://github.com/ramonfmir/lean-scheme/tree/submission`. As stated in the repository, the code runs on Lean 3.4.2 and mathlib's version `410ae5d` (March 26th 2019).

We assume some familiarity with basic commutative algebra. The definitions that we use are concisely presented in Chapter 1 of [24]. As a remark, all the rings that appear in this chapter are commutative rings with unity.

3.1 Commutative algebra

Before we delve into algebraic geometry, we discuss some notions in commutative algebra that are essential in scheme theory and that interact in a crucial way. We first develop the theory of ring localisations based on [1, 00CM]. We proceed by discussing briefly local rings [1, 07BH] and gluing functions [1, 00EI] and relating these concepts with the approach for ring localisation described in the previous part. Finally, we define the spectrum of a ring and prove a few important properties found in [1, 00DY].

3.1.1 Localisation

Localisation is a very powerful technique in commutative algebra that allows us to invert some elements of a ring or a module. This is used to display local properties of the object we are interested in. For example, if we are given the ring of continuous functions on a space X,

we might be interested in looking at its behaviour near a point $x \in X$. One thing we can do is add inverses to all the functions that do not vanish at x. By doing so, all of a sudden, the ones that do, which is the valuable data associated with x, can be manipulated in a natural and convenient way.

Normally, localisation is de ned as the following construction.

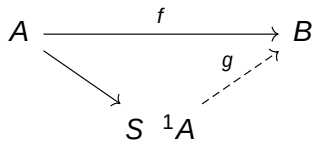
Definition 1. Given a ring R and a multiplicative subset $S \subseteq R$, the *localisation* of R with respect to S is $S^{-1}R = (R \times S)/\sim$ where $(a,s) \sim (b,t)$ if and only if there exists $u \in S$ such that $u(at - bs) = 0$. We denote by $\frac{a}{s}$ the equivalence class of (a,s). The operations are de ned in the obvious way [1, 00CM].

One can easily check that $\sim$ is an equivalence relation and that $S^{-1}R$ as above is in fact a ring. The de nition can be extended to modules, but we will ignore modules completely as basic scheme theory can be built solely with commutative rings.

There are two important kinds of multiplicative sets. Firstly, we can localise away from an element $f \in R$ letting $S = \{1, f, f^2, \ldots\}$, which will be denoted R_f. Secondly, given a prime ideal $\mathfrak{p} \subset R$, we can localise at $\mathfrak{p}$ letting $S = R \setminus \mathfrak{p}$, this will be denoted $R_{\mathfrak{p}}$.

As we will see shortly, this widely accepted de nition happens to be quite problematic to use directly in some places so we used an alternative equivalent de nition that simpli ed quite signi cantly some of our main proofs. In order to understand better where the new de nition comes from, we rst need to understand the *universal property of localisation*.

Proposition 2. Let A and B be rings and $f : A \to B$ a ring homomorphism. If for all $s \in S$, $f(s)$ is a unit in B, then there exists a unique ring homomorphism $g : S^{-1}A \to B$ such that the following diagram commutes.



The map $A \to S^{-1}A$ is the canonical map $a \mapsto \frac{a}{1}$ [1, 00CP].

Proof. We de ne $g\left(\frac{a}{s}\right) = \frac{f(a)}{f(s)}$. This is a well-de ned map as $f(s)$ is invertible by assumption and one checks that it inherits the homomorphism structure from f after some tedious calculations. Moreover, it is clear that $g\left(\frac{a}{1}\right) = f(a)$ so we have proved existence. For uniqueness, suppose that there is another map $h : S^{-1}A \to B$ such that $h\left(\frac{a}{1}\right) = f(a)$, then for $s \in S$ we would have $h\left(\frac{s}{1}\right) = f(s)$ and so $1 = h\left(\frac{s}{1}\right) h\left(\frac{1}{s}\right) = f(s)h\left(\frac{1}{s}\right)$. Therefore $h\left(\frac{1}{s}\right) = \frac{1}{f(s)}$ which implies that $h\left(\frac{a}{s}\right) = \frac{f(a)}{f(s)} = g\left(\frac{a}{s}\right)$. □

Thanks to this universal property, we can develop the theory of localised rings using the maps from the base ring. Suppose that we are given a ring B and a ring homomorphism $f : A \to B$ as above, then we know that there is a unique ring homomorphism $g : S^{-1}A \to B$. The question now is whether we can impose more conditions on f to ensure that g is an isomorphism. These conditions should completely characterise what we mean by localising

A with respect to S so that we can prove lemmas about this object (or rather this class of isomorphic objects) using them instead of the explicit construction. In January 2019, Neil Strickland came up with three necessary and su cient conditions that f has to satisfy for g to be an isomorphism.

Definition 3. Let A and B be rings, $S \subseteq A$ a multiplicative subset and $f : A \to B$ a ring homomorphism. We say that f satis es the *localisation predicate* of A localised with respect to S if the following conditions are satis ed:
(L1) For every $s \in S$, $f(s)$ has an inverse in B.
(L2) Elements of B have denominators of the form $f(s)$ for some $s \in S$.
(L3) Given $a \in A$, if $f(a) = 0$ then $as = 0$ for some $s \in S$.

```
variables {A : Type u} {B : Type v} [comm_ring A] [comm_ring B]
variables (S : set A) [is_submonoid S] (f : A → B) [is_ring_hom f]

def inverts_data :=  Π s : S, {b : B // (f s) * b = 1}

def has_denom_data :=  Π b : B, {sa : S × A // (f sa.1) * b = f sa.2}

def ann_aux := set.range (λ sa : {sa : S × A // sa.1 * sa.2 = 0}, sa.1.2)
def submonoid_ann : ideal A :=
⟨ann_aux S, ann_aux.zero S, ann_aux.add S, ann_aux.smul S⟩

structure is_localization_data :=
(inverts  : inverts_data S f)
(has_denom : has_denom_data S f)
(ker_le  : ker f ≤ submonoid_ann S)
```

It is worth commenting on the types of `inverts_data`, `has_denom_data` and `ann_aux`. They all use *subtypes*, introduced with the syntax `{x : T // p x}` where `p : T → Prop`. Subtypes are used to identify the values of a type satisfying a certain property.

The rst result that needs to be checked is that we can choose $B = S^{-1}A$ by which we mean the explicit construction from De nition 1.

Proposition 4. With the notation above, the canonical map $f : A \to S^{-1}A$ satis es the localisation predicate of A localised with respect to S.

```
lemma is_localization_data.of_of
: is_localization_data S (of : A → localization A S)
```

Proof. As expected, the proof is straightforward. Pick $s \in S$, then $f(s) = \frac{s}{1}$ has inverse $\frac{1}{s}$ hence $(L1)$ holds. For any $\frac{a}{s} \in S^{-1}A$ we have that $f(s)\frac{a}{s} = f(a)$, which is exactly $(L2)$. Finally, to show $(L3)$, suppose $\frac{a}{1} = \frac{0}{1}$ then, by de nition, there is $s \in S$ such that $sa = 0$. □

Next, we make sure that the predicate is invariant under isomorphisms.

Proposition 5. Let A, B and C be rings and $S \subseteq A$ a multiplicative set. If $f : A \to B$ a ring homomorphism satisfying the localisation predicate of A with respect to S and $g : B \to C$ is a ring isomorphism then $g \circ f : A \to C$ also satisfies the predicate of A localised with respect to S.

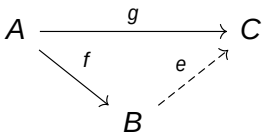
```
variables {C : Type w} [comm_ring C]
variables (S : set A) [is_submonoid S]
variables (f : A → B) [is_ring_hom f] (Hf : is_localization_data S f)
variables (g : B → C) (Hg₁ : function.bijective g) (Hg₂ : is_ring_hom g)

lemma is_localization_data.of_iso : is_localization_data S (g ∘ f)
```

Proof. Again, the axioms need to be checked. Pick $s \in S$. By assumption $f(s)$ has an inverse in B. The inverse for $g(f(s))$ is $g\left(\frac{1}{f(s)}\right)$ because $g(f(s))g\left(\frac{1}{f(s)}\right) = g\left(\frac{f(s)}{f(s)}\right) = g(1) = 1$ as g is a homomorphism so $(L1)$ follows. Now choose any $c \in C$. Since g is surjective, there exists $b \in B$ such that $g(b) = c$ and, by assumption, we can find $a \in A$ and $s \in S$ such that $f(s)b = f(a)$ hence $g(f(s))c = g(f(a))$, which shows $(L2)$. Suppose $g(f(a)) = 0 = g(0)$. Since g is injective $f(a) = 0$ so there is $s \in S$ such that $sa = 0$ as $(L3)$ requires. □

There is one sanity check missing to ensure that Strickland's axiomatisation is correct. If two rings B and C satisfy the localisation predicate for A localised at S we should be able to exhibit an isomorphism between them. It will follow from the universal property of the localisaiton predicate.

Proposition 6. Let A, B and C be rings and $f : A \to B$, $g : A \to C$ ring homomorphisms. If f satisfies the localisation property for A with respect to S and g satisfies $(L1)$ then there exists a unique ring homomorphism $e : B \to C$ such that the following diagram commutes.




```
variables (g : A → C) [is_ring_hom g] (Hg : inverts_data S g)

def is_localization_initial : B → C :=
  b, g (Hf.has_denom b).1.2 * Hg (Hf.has_denom b).1.1

instance : is_ring_hom (is_localization_initial S f Hf g Hg)

lemma is_localization_initial_comp (a : A)
: is_localization_initial S f Hf g Hg (f a) = g a

lemma is_localization_initial_unique (h : B → C) [is_ring_hom h] (b : B)
: is_localization_initial S f Hf (h ∘ f) (inverts_aux S f h Hf) b = h b
```

Proof. Choose $b\ 2\ B$ and write $b = \frac{f(a)}{f(s)}$ for $a\ 2\ A$ and $s\ 2\ S$, using $(L2)$. We de ne $e(b) = \frac{g(a)}{g(s)}$. Note that e is well-de ned as $g(s)$ is invertible by assumption. The most tedious part is proving that e is a ring homomorphism. Firstly, we claim that, for any $a_1; a_2\ 2\ A$, if $f(a_1) = f(a_2)$ then $g(a_1) = g(a_2)$. By $(L3)$, we have that $sa_1 = sa_2$ for some $s\ 2\ S$. But then $g(s)g(a_1) = g(s)g(a_2)$ and, since $g(s)$ is invertible by assumption, $g(a_1) = g(a_2)$. Let $x; y\ 2\ B$. Then, by $(L2)$ in B, $x = \frac{f(a_1)}{f(s_1)}$, $y = \frac{f(a_2)}{f(s_2)}$ and $xy = \frac{f(a_3)}{f(s_3)}$ for some $a_1; a_2; a_3\ 2\ A$ and $s_1; s_2; s_3\ 2\ S$. Hence $f(a_3 s_1 s_2) = f(a_1 a_2 s_3)$ and thus, by the claim, $g(a_3 s_1 s_2) = g(a_1 a_2 s_3)$ so $\frac{g(a_3)}{g(s_3)} = \frac{g(a_1)}{g(s_1)}\frac{g(a_2)}{g(s_2)}$ which means that $e(xy) = e(x)e(y)$. The same argument works for addition. We conclude that e is a ring homomorphism. Now take any $a\ 2\ A$. We have $f(a) = \frac{f(a_1)}{f(s_1)}$ for some $a_1\ 2\ A$ and $s_1\ 2\ S$ and, by de nition, $e(f(a)) = \frac{g(a_1)}{g(s_1)}$. It remains to show that $g(a) = \frac{g(a_1)}{g(s_1)}$ but $f(s_1 a) = f(a_1)$ so from the claim it follows that $g(s_1 a) = g(a_1)$ which implies that $e \quad f = g$. Finally, for uniqueness suppose $h : B \ ! \ C$ is any ring homomorphism. We show that if $h \quad f = g$ then $h = e$. Choose $b\ 2\ B$ and write $b = \frac{f(a)}{f(s)}$ for some $a\ 2\ A$ and $s\ 2\ S$ using $(L2)$. Now $e(b) = \frac{g(a)}{g(s)} = \frac{h(f(a))}{h(f(s))} = h(b)$. □

We now show that e is bijective if g also satis es the localisation property for A with respect to S, which implies the correctness of the predicate. With the same setting as above, pick any $c\ 2\ C$. Then $(L2)$ tells us that $c = \frac{g(a)}{g(s)}$ for some $s\ 2\ S$ and $a\ 2\ A$. Choose $b = \frac{f(a)}{f(s)}\ 2\ B$. Then, by the universal property $e(b) = c$ so, in particular, e is surjective. Now suppose that $e(b) = 0$ for some $b\ 2\ B$. Again by $(L2)$, we nd $a\ 2\ A$ and $s\ 2\ S$ such that $f(s)b = f(a)$ and therefore, by the universal property, $g(a) = 0$. We use $(L3)$ to obtain that $ta = 0$ in A for some $t\ 2\ S$. But then $f(ts)b = f(ta) = 0$ in B and $ts\ 2\ S$ so $f(ts)$ is invertible and therefore $b = 0$ hence e is also injective.

This approach has many advantages. For instance, for a ring R and $f; g\ 2\ R$ we know that R_{fg} and $(R_f)_{\frac{g}{1}}$ are isomorphic. Unfortunately, they are not de nitionally equal as types in Lean. For Lean, elements of the former are equivalence classes of pairs in R and elements of the latter are equivalence classes of pairs of equivalence classes of pairs in R so, in a sense, it is not surprising that Lean thinks that they are completely di erent.

If we have a lemma about R_{fg} de ned as in De nition 1 then we cannot apply it directly to $(R_f)_{\frac{g}{1}}$. Instead, one would have to explicitly work with the isomorphism, which can unnecessarily complicate the problem. This was, in fact, a major issue in the rst formalisation attempt. The way to solve it is to prove such lemma for an arbitrary ring satisfying the localisation predicate for R away from fg and then prove that $(R_f)_{\frac{g}{1}}$ satis es it. This was one of the tests proposed by Strickland to ensure that the predicate was properly de ned. We successfully managed to prove it shortly after he proposed it.

Proposition 7. The ring $(R_f)_{\frac{g}{1}}$ satis es the localisation predicate of R away from fg.

```
parameters {R : Type u} [comm_ring R] parameters (f g : R)

def Rfg := localization (localization R (powers f)) (powers (of g))
lemma is_localization_data_away_away
: is_localization_data (powers (f * g)) ((of   of) : R  !  Rfg)
```

Proof. Firstly we de ne $h : R \; ! \; (R_f)_{\frac{g}{1}}$ as $h(x) = \frac{x=1}{1=1}$. It is an easy exercise to check that h is a ring homomorphism. We proceed by checking the three properties. For $(L1)$, consider $h(f^n g^n) = \frac{f^n g^n=1}{1=1}$, it su ces to check that $\frac{1=f^n}{g^n=1}$ 2 $(R_f)_{\frac{g}{1}}$ is its inverse. Note that any element in $(R_f)_{\frac{g}{1}}$ is of the form $\frac{a=f^n}{g^m=1}$. Let $N = max fn; mg$. Then multiplying through by $f(f^N g^N)$ clears the denominators, which is exactly what we need to show $(L2)$. Finally, we prove $(L3)$. If $h(a) = \frac{a=1}{1=1} = \frac{0=1}{1=1}$, by de nition there exists n and m such that $f^n g^m a = 0$. Again, we let $N = max fn; mg$ and observe that $f^N g^N a = 0$ as required. □

These sort of rather elementary arguments appear frequently throughout the codebase. Another bene t of this approach is that they can be easily separated from the main proofs so that no focus from the core arguments is lost.

As a concluding remark, we point out that the localisation predicate `is_localization_data` is purely constructive, it carries data instead of statements. Observe that instead of asserting the existence of an inverse it requires the user to provide one. However, in many cases it is convenient to work with non-constructive predicates and they are also de ned together with a simple interface to go from one to the other.

```
def inverts (S : set A) (f : A ! B) : Prop :=
8 s : S, 9 si : B, (f s) * si = 1

def has_denom (S : set A) (f : A ! B) : Prop :=
8 b : B, 9 (sa : S   A), (f sa.1) * b = (f sa.2)

def is_localization : Prop :=
(inverts S f) ^ (has_denom S f) ^ (ker f   submonoid_ann S)
```

3.1.2 Local rings

In this subsection we brie y discuss local rings. They play an important role when describing the local behaviour of algebraic structures. In particular, we will impose locality, unsurprisingly, in the de nition of locally ringed spaces in section 3.3.

Definition 8. We say that a ring R is a *local ring* if it has a unique maximal ideal $m \subset R$.

Note that not all localised rings are local rings. For instance $C[x]$ localised away from x has in nitely many maximal ideals, any ideal of the form $\left(\frac{x}{1}\right)$ with $\not\in 0$ is maximal. On the other hand, Z localised at (3) has exactly one maximal ideal, namely $\left(\frac{3}{1}\right)$. This observation leads us to the following result.

Lemma 9. If $p \subset R$ is a prime ideal, R_p is a local ring.

```
variables {P : ideal A} (HP : ideal.is_prime P)
variables (Hloc : is_localization_data (-P : set A) f)
```

```
lemma is_local_ring.of_is_localization_away_from_prime : is_local_ring B
```

Proof. To prove that $R_\mathfrak{p}$ is local it su ces to prove that the nonunits $I = R_\mathfrak{p} \setminus R_\mathfrak{p}$ form an ideal. Formally, rst one needs to check that $R_\mathfrak{p}$ is not the zero ring. Suppose $0 = 1$, then $1 \in \ker(f)$ so by $(L3)$ there is a $v \notin \mathfrak{p}$ such that $1v = 0$ but $0 \in \mathfrak{p}$ for any prime ideal, so this is nonsense. Now take $x, y \in I$. We need to show that $x + y \in I$. Assume for a contradiction that $(x + y)z = 1$ for some $z \in R_\mathfrak{p}$. Using $(L2)$ we write $x = \frac{f(a)}{f(r)}, y = \frac{f(b)}{f(s)}$ and $z = \frac{f(c)}{f(t)}$ for $r, s, t \notin \mathfrak{p}$; $a, b \in \mathfrak{p}$ and $c \in R$. By $(L3)$, we have that there exists $v \notin \mathfrak{p}$ such that $v(acs + bcr \quad rst) = 0 \in \mathfrak{p}$ but since $acs \in \mathfrak{p}$ and $bcr \in \mathfrak{p}$ we must have that $vrst \in \mathfrak{p}$ which is a contradiction. □

3.1.3 Gluing functions

There is one result in this section and, for now, we will deal with it purely algebraically. In section 3.4 we will see how it is much deeper and we will rigorously deal with its geometric implications. Intuitively, one can think of elements of R_f as functions whose domain are those points that do not vanish at f. If two functions coincide in the intersection of their domains then we can glue them together.

Lemma 10. Let R be a ring and $f_1, \ldots, f_n \in R$ with the property that $(f_1, \ldots, f_n) = R$. Then the short sequence:

$$0 \quad ! \quad R \quad ! \quad \bigoplus_i^n R_{f_i} \quad ! \quad \bigoplus_{i,j}^n R_{f_i f_j}$$

with $(x) = \left(\frac{x}{1}, \ldots, \frac{x}{1}\right)$ and $\left(\frac{x_1}{f_1^{l_1}}, \ldots, \frac{x_n}{f_n^{l_n}}\right) = \left(\ldots, \frac{x_i}{f_i^{l_i}} \quad \frac{x_j}{f_j^{l_j}}, \ldots\right)$ is exact [1, 00EJ].

```
variables {R : Type u} [comm_ring R]
variables {  : Type v} [fintype   ] {f :    !  R}

variables {Rfi :    !  Type w} [   i, comm_ring (Rfi i)]
variables { i :    i, R  !  (Rfi i)} [   i, is_ring_hom ( i i)]

variables {Rfij :    !    !  Type w} [   i j, comm_ring (Rfij i j)]
variables { ij :    i j, R  !  (Rfij i j)} [   i j, is_ring_hom ( ij i j)]

variables
    (H  :    i, is_localization_data (powers (f i)) ( i i))
    (H  :    i j, is_localization_data (powers ((f i)*(f j))) ( ij i j))
    (H1 : (1 : R) ∈ ideal.span (set.range f))

def    : R  !    i, Rfi i :=    r i, ( i i) r
```

```
def  1 : (  i, Rfi i) ! (  i j, Rfij i j)
:=   ri i j, (is_localization_initial
    (powers (f j)) ( i j) (H  j) ( ij i j) (inverts_fj H  i j)) (ri j)

def  2 : (  i, Rfi i) ! (  i j, Rfij i j)
:=   ri i j, (is_localization_initial
    (powers (f i)) ( i i) (H  i) ( ij i j) (inverts_fi H  i j)) (ri i)

def   : (  i, Rfi i) ! (  i j, Rfij i j) :=   r, ( 1 H  r) - ( 2 H  r)

lemma standard_covering₁ : function.injective
lemma standard_covering₂ : (  H ) s = 0 $ 9 r : R,   r = s
```

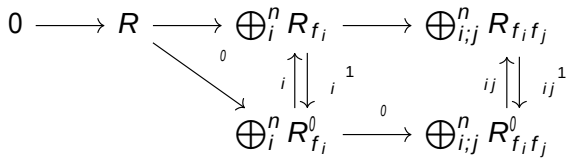
Proof. Suppose that $(x) = \left(\frac{0}{1};:::;\frac{0}{1}\right)$ for some $x\ 2\ R$. Then $\frac{x}{1} = \frac{0}{1}$ in every R_{f_i}. By $(L3)$, for any i we have that $f_i^{n_i}x = 0$ for some n_i. We know that $1 = \sum_i a_i f_i$ for some $a_i\ 2\ R$. It su ces to choose $M = \sum_i n_i$ and deduce that $x = 1^M x = \left(\sum_i a_i f_i\right)^M x = 0$. The last equality is justi ed by the multinomial theorem and the pigeonhole principle. This concludes the rst part of the proof.

For the second part, rstly note that $im(\ )\quad \ker(\ )$ is obvious because for any $x\ 2\ R$, $(\ (x)) = \left(\frac{x}{1};:::;\frac{x}{1}\right) = \left(:::;\frac{x}{1}\quad\frac{x}{1};:::\right) = \left(\frac{0}{1};:::;\frac{0}{1}\right)$. The other direction is the core of this lemma. What we need to show is that if $\left(\frac{x_1}{f_1^{r_1}};:::;\frac{x_n}{f_n^{r_n}}\right) = \left(\frac{0}{1};:::;\frac{0}{1}\right)$ then we can nd $x\ 2\ R$ such that $\frac{x}{1} = \frac{x_i}{f_i^{r_i}}$ for all i. So assume that $\frac{x_i}{f_i^{r_i}}\quad\frac{x_j}{f_j^{r_j}} = 0$ in $R_{f_if_j}$ for all i and j. By $(L3)$ this translates to $f_i^{n_{ij}}f_j^{n_{ij}}(f_j^{r_j}x_i\quad f_i^{r_i}x_j) = 0$ for some n_{ij}. Choose $M = max\{n_{ij}\}$ so that $f_i^M f_j^M(f_j^{r_j}x_i\quad f_i^{r_i}x_j) = 0$ for all i and j or, more conveniently, $f_j^{M+r_j}f_i^M x_i \overset{(y)}{=} f_i^{M+r_i}f_j^M x_j$. Note that $(f_1^{M+r_1};:::;f_n^{M+r_n}) = R$ so $1 = \sum_i a_i f_i^{M+r_i}$ for some $a_i\ 2\ R$. Therefore:

$$\frac{x_i}{f_i^{r_i}} = \frac{f_i^M x_i}{f_i^{M+r_i}} = \sum_j \frac{a_j f_j^{M+r_j} f_i^M x_i}{f_i^{M+r_i}} \overset{(y)}{=} \sum_j \frac{a_j f_i^{M+r_i} f_j^M x_j}{f_i^{M+r_i}} = \frac{\sum_j a_j f_j^M x_j}{1}:$$

The element we needed to nd is precisely $\sum_j a_j f_j^M x_j$. □

This result is a good example of the usefulness of the localisation predicate. In the rst attempt of formalising it, the authors used the explicit construction as a quotient. The problem was that the objects they had, which we will encounter later, where only canonically isomorphic to R_{f_i} and $R_{f_if_j}$. The situation was the following:



This means that one needs to explicitly check that all the diagrams commute in order to use the exact sequence as needed. Moreover, these maps, although evident from a mathe-

matical point of view, were quite tedious to manipulate in Lean. With our new approach, whenever we encounter these canonically isomorphic rings, we will simply prove the localisation predicate hence avoiding any unnecessarily complicated argument about canonical isomorphisms.

3.1.4 The spectrum of a ring

In this subsection we develop the main tool to connect algebra and geometry. For now the goal is to construct a topological space associated to a given ring R and to prove some basic properties about it. This topological space is precisely $Spec(R)$, which we mentioned in section 1.2. Later on, we will see how this construction, which at rst might seem arbitrary, captures very nicely the behaviour of R seen as a geometric object.

Definition 11. Given a ring R, its *spectrum* is the set of all its prime ideals and is denoted by $Spec(R)$. For T R, we de ne $V(T) = fp\ 2\ Spec(R)\ j\ f\ 2\ p$ for all $f\ 2\ Tg$ and $D(T)$ to be the complement of $V(T)$ [1, 00DZ].

```
def Spec := {P : ideal R // ideal.is_prime P}
def V : set R ! set (Spec R) :=  E, {P | E  P.val}
def V' : R ! set (Spec R) :=  f, {P | f 2 P.val}
def D : set R ! set (Spec R) :=  E, -V(E)
def D' : R ! set (Spec R) :=  f, -V'(f)
```

The next step is to de ne a topology on $Spec(R)$ but, before that, we prove a few basic properties that will become useful later on.

Lemma 12. Given a ring R we have:
(1) For S R, $V(S) = V((S))$.
(2) For I C R, $V(I) = ?$ if and only if $I = R$.
(3) For $f;g\ 2\ R$, $D(fg) = D(f) \setminus D(g)$.
(4) For $f_i\ 2\ R$, $\bigcup D(f_i) = Spec(R)\ n\ V(ff_ig)$ [1, 00E0].

```
lemma Spec.V.set_eq_span (S : set R) : V(S) = V(ideal.span S)
lemma Spec.V.empty_iff_ideal_top (I : ideal R) : V(I.val) = ? $ I = >
lemma Spec.D'.product_eq_inter (f g : R) : D'(f * g) = D'(f) \ D'(g)
lemma Spec.D'.union (F : set R) : ⋃₀ (D' '' F) = -V(F)
```

Proof. If p $2\ V(S)$ then S p and by basic properties of ideals (S) p hence p $2\ V((S))$. The other direction is obvious as S (S). This shows (1). For (2), note that $V(I) = ?$ means that I is not contained in any prime ideal and in particular it is not contained in any maximal ideal, hence $I = R$ by Zorn's lemma. Conversely, if $I = R$, it is clear that no prime ideal contains it. Part (3) follows from the de nition of a prime ideal: fg 2 p if and only if f 2 p and g 2 p. Finally, we show (4). If p $2\ D(f_i)$, then f_i 2 p so p 2 $V(ff_ig)$ for all i. On the other hand, if p 2 $V(ff_ig)$, clearly f_i 2 p for some i. □

There are two subtleties worth mentioning about these lemmas. In (2), we write `V(I.val)` instead of `V(I)`. Lean allows coercions between types, and ideals are coerced to sets as one would expect. It often happens that some function takes a set as an input but, if given an ideal instead, Lean will automatically infer this coercion. In this case, however, since the right hand side of the equality is `?`, whose type could be `set A` for any `A`, the elaborator cannot deduce that the argument expected by `V` has to be of type `set R` and so it cannot perform the coercion. To make this information available to the elaborator, we would need to write `(? : set (Spec R)) = V(I)`. It is a minor issue but it illustrates the amount of reasoning behind every small design decision and the sort of knowledge about what happens in the background to properly understand the code.

The other observation is that we write `I = >` meaning $I = R$. Type-theoretically it would not make sense to write `I = R`. In mathlib this is dealt with by proving that the ideals of a ring form a complete lattice, where `>` is an upper bound for every ideal so it is all of R. We will not go into much more detail but this is yet another example of the paradigm shift necessary to do formal mathematics.

Going back to $Spec(R)$, we now give it the desired structure.

Lemma 13 (Zariski topology)**.** The set $Spec(R)$ has the structure of a topological space where the closed subsets are the subsets of the form $V(T)$ for $T \quad R$ [1, 00E1].

```
instance (R : Type u) [comm_ring R] : topological_space (Spec R)
```

Proof. We need to check the axioms. It follows directly from Lemma 12 (2) that $V(R) = ?$. We show that $V(I) \; [\; V(J) = V(I \setminus J)$ for $I; J \subset R$. We are allowed to only consider ideals because of Lemma 12 (1). If $I \quad$ p or $J \quad$ p clearly $I \setminus J \quad$ p. Conversely, if $I \setminus J \quad$ p then $IJ \quad$ p so either $I \quad$ p or $J \quad$ p as required because p is prime. Finally, we prove that $\bigcap V(I_i) = V(\bigcup I_i)$. But p 2 $V(I_i)$ for all i translates to $I_i \quad$ p for all i which implies p 2 $V(\bigcup I_i)$. Conversely, if $\bigcup I_i \quad$ p clearly $I_i \quad$ p for all i. □

Observe that the proofs of the rst two axioms above are, in essence, the same as the proofs of Lemma 12 (3) and (4). The reason why we have proved results about D applied to a single element separately is because these open sets play an important role in the de nition of a scheme. Opens of the form $D(f)$ are called *standard opens*.

Lemma 14. Let R be a ring. The set $B = \{D(f) \mid f \in R\}$ is a basis for $Spec(R)$. Furthermore, B is a *standard basis* meaning that $Spec(R) \in B$ and that if $U, V \in B$ then $U \setminus V \in B$ [1, 00DY].

```
def D_fs := {U : opens (Spec R) | ∃ f : R, U = D0(f)}

lemma D_fs_basis : opens.is_basis D_fs
lemma D_fs_standard_basis :
opens.univ ∈ D_fs ∧ ∀ U ∈ D_fs → V ∈ D_fs → U ∩ V ∈ D_fs
```

Proof. Pick any p 2 *Spec*(*R*) and an open set *U* *Spec*(*R*) such that p 2 *U*. Note that by de nition of the Zariski topology, *U* = *D*(*T*) for some *T* *R*, hence *T* * p so in particular there is some *f* 2 *T* such that *f* 2 p hence p 2 *D*(*f*). Moreover, it is clear that *D*(*f*) *D*(*T*) because if *f* 2 q then *T* * q hence we have proved that the standard opens form a basis.

For the last part, certainly *D*(1) = *Spec*(*R*) as 1 2 p for all prime ideals p C *R* and the fact that intersections of basis elements are in the basis is exactly Lemma 12 (3). □

We now deduce an important property about the standard basis and use it to prove compactness of *Spec*(*R*).

Lemma 15. For any set *U* = *fD*(*f*) *j f* 2 *S* *Rg* such that ⋃*U* = *Spec*(*R*) there exists a nite subset *U*′ *U* such that ⋃*U*′ = *Spec*(*R*) [1, 00E8].

```
lemma D_fs_quasi_compact :
∀ S : set R, ⋃₀ (Spec.D' '' S) = Spec.univ R →
∃ F ⊆ S,
  set.finite F
  ∧ ⋃₀ (Spec.D' '' F) = Spec.univ R
```

Proof. From Lemma 12 (4) we deduce that *V*(*S*) = ? and hence by Lemma 12 (1) and (2), (*S*) = *R*. Therefore 1 = $\sum_{i=1}^{n} r_i f_i$ for r_i 2 *R* and f_i 2 *S*. Choose *F* = *ff*$_1$;:::;f_n*g*. It just remains to show that ⋃*D*(f_i) = *Spec*(*R*). Again, by Lemma 12 (4) we just need to prove that *V*(*F*) = ?, which is obvious because 1 2 (*F*). □

Corollary 16. The space *Spec*(*R*) is compact [1, 00E8].

```
lemma Spec.compact : compact_space (Spec R)
```

Proof. Suppose *U* = *fU*$_i$*g* is an open cover of *Spec*(*R*), i.e. ⋃*U* = *Spec*(*R*). By standard topology, we can re ne the cover with basis elements so there exists *S* *R* such that *U*′ = *fD*(f_i) *j* f_i 2 *Sg* is an open cover and moreover *D*(f_i) U_i for all *i*. It follows from 15 that, after possibly renaming, we can nd *ff*$_1$;:::;f_n*g* *S* such that *D*(f_0)[[*D*(f_n) = *Spec*(*R*). Therefore *fU*$_1$;:::;U_n*g* is a nite subcover of *U* so *Spec*(*R*) is compact. □

So far we have seen that *Spec*(*R*) is a topological space associated with a particular basis and that is compact. The next step is to study the maps between topological spaces of this form given by maps between the underlying rings.

Definition 17. Let *A* and *B* be rings and *f* : *A* ! *B* a ring homomorphism. Then there is an *induced map* : *Spec*(*B*) ! *Spec*(*A*) given by the rule (p) = *f* ^{1}(p) [1, 00E2].

```
def Zariski.induced : Spec B → Spec A :=
  ⟨P, HPi, ideal.comap f P, @ideal.is_prime.comap _ _ _ _ f _ P HPi⟩
```

Note that the de nition above includes the justi cation that the map is well-de ned, which is the case because if p is prime then so is $f^{-1}(\mathfrak{p})$. The purpose of the @ is to make implicit arguments explicit. Sometimes, the Lean elaborator cannot infer these arguments from the context and they need to be provided manually. In this case, Lean could not deduce that the ideal p (P) was prime (HP) so we had to pass it as an argument.

After this aside, we proceed with our study of these maps.

Lemma 18. The map $\varphi : Spec(B) \to Spec(A)$ as above is continuous [1, 00E2].

```
lemma Zariski.induced.continuous : continuous (Zariski.induced f)
```

Proof. Suppose $U \subseteq Spec(B)$ is an open set or, in other words, $U = D(S)$ for some $S \subseteq R$. We claim that $\varphi^{-1}(U) = D(f(S))$, which implies the result. Let $\mathfrak{p} \subset B$ be a prime ideal. Then $\mathfrak{p} \in \varphi^{-1}(U)$ means that $\varphi(\mathfrak{p}) \in D(S)$ and, by de nition, $S \not\subseteq \varphi^{-1}(\mathfrak{p})$ which is equivalent to $f(S) \not\subseteq \mathfrak{p}$ and this translates to $\mathfrak{p} \in D(f(S))$ as required. □

Next, we give ring localisations a geometric interpretation. Let R be a ring and $f \in R$ and let $h : R \to R_f$ be a ring homomorphism satisfying the localisation predicate for R away from f as introduced in section 3.1.1. It induces a continuous map $\varphi : Spec(R_f) \to Spec(R)$.

```
variables {R : Type u} [comm_ring R]
variables {Rf : Type u} [comm_ring Rf] {h : R → Rf} [is_ring_hom h]
variables {f : R} (HL : is_localization_data (powers f) h)

def φ : Spec Rf → Spec R := Zariski.induced h
```

It turns out that φ induces a homeomorphism between $D(f)$ and $Spec(R_f)$. We will prove this fact in several steps.

Lemma 19. The map $\varphi : Spec(R_f) \to Spec(R)$ is injective [1, 00E4].

```
lemma phi_injective : function.injective (φ HL)
```

Proof. Let $\mathfrak{p}, \mathfrak{q} \subset R_f$ and suppose that $\varphi(\mathfrak{p}) = \varphi(\mathfrak{q})$. We show that $\mathfrak{p} \subseteq \mathfrak{q}$. Pick $x \in \mathfrak{p}$ then, by $(L2)$, there is $a \in R$ and $n \in \mathbb{N}$ such that $h(f^n)x = h(a)$ but then $h(a) \in \mathfrak{p}$ so $a \in \varphi(\mathfrak{p}) = \varphi(\mathfrak{q})$ and, hence, we have that $h(a) \in \mathfrak{q}$. Since it is prime, if we prove that $h(f^n) \notin \mathfrak{q}$ then $x \in \mathfrak{q}$ will follow. But assume that $h(f^n) \in \mathfrak{q}$, then since by $(L1)$ it is invertible we would have $1 \in \mathfrak{q}$ which is absurd. The other direction is identical. □

Lemma 20. The map $\varphi : Spec(R_f) \to Spec(R)$ is an open map [1, 00E4].

```
lemma phi_opens : ∀ U : set (Spec Rf), is_open U → is_open ((φ HL) '' U)
```

Proof. It su ces to show that if $U \subseteq \mathrm{Spec}(R_f)$ is open then $\varphi(U) \subseteq \mathrm{Spec}(R)$ is open, the other implication follows from continuity and injectivity of φ.

First, we claim that for a prime $\mathfrak{p} \subset R$, if $f \notin \mathfrak{p}$ then $(h(\mathfrak{p})) \subset R_f$ is prime. We see that $1 \notin (h(\mathfrak{p}))$ because otherwise there is some $x \in \mathfrak{p}$ and $y \in R_f$ such that $yh(x) = 1 = h(1)$. Then, by $(L2)$, $y = \frac{h(a)}{h(f^n)}$ for some n so $h(xa) = h(f^n)$ and $f^m xa = f^{m+n}$ for some m by $(L3)$. But then $f^{m+n} \in \mathfrak{p}$ so $f \in \mathfrak{p}$ as it is prime, which is a contradiction. Now take $x, y \in R_f$, write $x = \frac{h(a)}{h(f^n)}$, $y = \frac{h(b)}{h(f^m)}$ and suppose that $\frac{h(a)}{h(f^n)}\frac{h(b)}{h(f^m)} \in (h(\mathfrak{p}))$. We have that $\frac{h(a)}{h(f^n)}\frac{h(b)}{h(f^m)} = \frac{h(c)}{h(f^k)}$ for some $c \in \mathfrak{p}$. Using $(L3)$ we have $f^l(f^k ab - cf^{n+m}) = 0$ and hence $f^{l+k}ab \in \mathfrak{p}$ but $f \notin \mathfrak{p}$ so either $a \in \mathfrak{p}$ or $b \in \mathfrak{p}$. Without loss of generality, suppose $a \in \mathfrak{p}$. Then $\frac{h(a)}{h(f^n)} \in (h(\mathfrak{p}))$ as required so $(h(\mathfrak{p}))$ is prime. In addition, $\varphi((h(\mathfrak{p}))) = \mathfrak{p}$. If $x \in \mathfrak{p}$ then $h(x) \in h(\mathfrak{p}) \subseteq (h(\mathfrak{p}))$ so obviously $x \in \varphi((h(\mathfrak{p})))$. Conversely, if $x \in \varphi((h(\mathfrak{p})))$ then $h(x) = \frac{h(a)}{h(f^n)}$ for some $a \in \mathfrak{p}$ so we have $f^m(f^n x - a) = 0$ and, by the same argument as before, we deduce that $x \in \mathfrak{p}$.

Going back to the main proof, if U is open then $U = D(S)$ for some $S \subseteq R_f$. We claim that $\varphi(D(S)) = D(T)$ where $T = (f)(h^{-1}(S)) \subseteq R$.

Suppose that $\mathfrak{p} \in D(T) \subseteq \mathrm{Spec}(R)$, i.e. $T \not\subseteq \mathfrak{p}$. Then there is $x \in h^{-1}(S)$ and n such that $f^n x \notin \mathfrak{p}$ so, in particular, $f \notin \mathfrak{p}$. By the previous result, $\mathfrak{q} = (h(\mathfrak{p})) \subset R_f$ is prime. Moreover $\varphi(\mathfrak{q}) = \mathfrak{p}$ so it is enough to show that $\mathfrak{q} \in D(S) \subseteq \mathrm{Spec}(R_f)$. We have $h(f^n x) \notin \mathfrak{q}$ and hence $h(x) \notin \mathfrak{q}$ but $h(x) \in S$ so $S \not\subseteq \mathfrak{q}$ as required.

Conversely, assume $\mathfrak{p} \in \varphi(D(S))$ then $\mathfrak{p} = \varphi(\mathfrak{q})$ for some $\mathfrak{q} \in D(S)$. The first observation is that $f \notin \mathfrak{p}$. If it were, then $h(f) \in \mathfrak{q}$ so, arguing similarly as the previous case, $1 \in \mathfrak{q}$ which is a contradiction. Therefore, we can apply the claim to obtain that $\mathfrak{p} = \varphi((h(\mathfrak{p})))$ and, since φ is injective, $\mathfrak{q} = (h(\mathfrak{p}))$. Hence there is some $s \in S$ such that $s \notin (h(\mathfrak{p}))$. Write $s = \frac{h(a)}{h(f^n)}$. Then $a \in h^{-1}(S)$ and we must have $a \notin \mathfrak{p}$. Since $\mathfrak{p}$ is prime, $fa \notin \mathfrak{p}$ so $T \not\subseteq \mathfrak{p}$. □

Lemma 21. The image of $\varphi : \mathrm{Spec}(R_f) \to \mathrm{Spec}(R)$ is $D(f)$ [1, 00E4].

```
lemma phi_image_Df : ( HL) '' Spec.univ Rf = Spec.D'(f)
```

Proof. It follows form the proof of Lemma 20 when $S = \{f\}$. □

As we will see, the fact that we can think of $D(f)$ as $\mathrm{Spec}(R_f)$ is crucial so that schemes make sense. In particular, we use it to prove the following result.

Lemma 22. For any f and any open cover $\{D(g_i)\}$ of $D(f)$, we can find a finite subcover $D(f) = D(g_1) \cup \cdots \cup D(g_n)$ [1, 04PM].

```
def basis_elements_finite_subcover : Prop :=
∀ {U} (BU : U ⊆ B) (OC : covering_standard_basis B U),
∃ (ι : Type u) (Hfin : fintype ι) (f : ι → OC.ι), ⋃( (OC.Uis ∘ f)) = U

theorem structure_presheaf.standard_opens_finite_subcover
: basis_elements_finite_subcover (D_fs_standard_basis R)
```

Proof. We just saw that $D(f) = Spec(R_f)$ so by compactness (Corollary 16) we can choose a nite subcover. More precisely, we can consider the cover $f \ ^1(D(g_i))g$ of $Spec(R_f)$. Then we can choose $f \ ^1(D(g_1)); : : : ; \ ^1(D(g_n))g$ such that they cover $Spec(R_f)$. Recall from the proof of Lemma 18 that these are actually $fD(h(g_1)); : : : ; D(h(g_n))g$. Clearly $f \ (D(h(g_1))); : : : ; \ (D(h(g_n)))g$ covers $D(f)$. It remains to show that $(D(h(g_i))) \quad D(g_i)$ and then we can choose $fD(g_1); : : : ; D(g_n)g$. Suppose p $2 \ (D(h(g_i)))$ then p = (q) for some q $2\ D(h(g_i))$ and hence $h(g_i) \ 2$ q so clearly $g_i \ 2$ p and thus p $2\ D(g_i)$. □

We use the properties of again to prove a result that we will use later on to show that if $D(g) \quad D(f)$ then there is a map $R_f \ ! \ R_g$.

Lemma 23. For $f; g \ 2 \ R$, if $D(g) \quad D(f)$ then f is invertible in R_g [1, 01HS].

```
lemma inverts.of_Dfs_subset {f g : R} (H : D'(g)   D'(f))
: inverts (powers f) (of : R ! localization R (powers g))
```

Proof. Let $\frac{f^n}{1} \ 2 \ R_g$. If $\frac{f^n}{1} \ 2$ p C R_g prime then $f \ 2$ q = (p) but q $2\ D(g)$ by Lemma 21 so q $2\ D(f)$ which implies that $f \ 2$ q and that is a contradiction. Since $\frac{f^n}{1}$ is not contained in any prime ideal of R_g, it is a unit in R_g. □

Moreover, we can say something about the relationship between f and g as elements.

Lemma 24. Let $f; g \ 2 \ R$ and suppose $D(g) \quad D(f)$. Then $g^e = af$ for some e and some $a \ 2 \ R$ [1, 01HS].

```
lemma pow_eq.of_Dfs_subset {f g : R} (H : D'(g)   D'(f))
: 9 (a : R) (e : N), g^e = a * f
```

Proof. From the previous lemma, there exists $r \ 2 \ R$ and n such that $\frac{f}{1}\frac{r}{g^n} = \frac{1}{1}$. Therefore, by $(L3)$, there exists m such that $g^m(rf \quad g^n) = 0$. Choosing $a = g^m r$ and $e = n + m$ we obtain $g^e = af$ as required. □

3.2 Sheaf theory

In the previous section we gave some geometric intuition of the algebraic objects de ned. We could not be very precise because we were missing an important bit of machinery, which will be developed in the following few subsections. Given a topological space X we would like to think of `functions' de ned on $X$ in a way such that it still makes sense to talk about them when restricted to some open set $U \quad X$. With some extra conditions, we will see how we can furthermore use this structure to make sense of the local behaviour of these `functions'.

We begin by de ning presheaves [1, 006D], sheaves [1, 006S] and stalks [1, 0078]. Then we discuss the slightly more technical issue of extending a presheaf de ned on a basis to a presheaf de ned on the whole space [1, 009H].

3.2.1 Presheaves

The first structure we need is the one capturing the notion of restriction.

Definition 25. Let X be a topological space. A *presheaf of sets* F is given by the rules:
(a) For every open subset $U \subseteq X$, there is a set $F(U)$. The elements of this set are called the *sections* of F over U.
(b) If $U, V \subseteq X$ are open subsets such that $V \subseteq U$, there is a map $\rho_{UV} : F(U) \to F(V)$, called a *restriction map*.
(c) ρ_{UU} is the identity map.
(d) If $U, V, W \subseteq X$ are open subsets such that $W \subseteq V \subseteq U$, then $\rho_{UW} = \rho_{VW} \circ \rho_{UV}$.
The set $F(U)$ is also sometimes denoted $\Gamma(U, F)$ [1, 006E].

```
structure presheaf (X : Type u) [topological_space X] :=
(F     : opens X → Type v)
(res   : ∀ (U V) (HVU : V ⊆ U), F U → F V)
(Hid   : ∀ (U), res U U (set.subset.refl U) = id)
(Hcomp : ∀ (U V W) (HWV : W ⊆ V) (HVU : V ⊆ U),
  res U W (set.subset.trans HWV HVU) = res V W HWV ∘ res U V HVU)
```

In the language of category theory, a presheaf on a topological space X is a contravariant functor from the category of open sets of X, with morphisms given by inclusion, to the category of sets, groups or rings (or any small category). This was one of our initial design decisions. We were aware of the benefits of a categorical treatment. However, it is one of the most difficult areas to formalise because of its foundational nature and at the start of the project there were still a few important definitions missing so we decided to not use the category_theory library at all at the cost of some generality.

At the time of writing the situation is very different. Scott Morrison, Mario Carneiro and Reid Barton have given the categorical definition of a presheaf. As mentioned in section 1.4, our definitions have been used in the Perfectoid Spaces project and are mathematically correct. However, we think that eventually they should be replaced by the ones in mathlib.

We continue our development by looking at maps between presheaves.

Definition 26. Let F and G be presheaves of sets on a topological space X. A *morphism of presheaves of sets* $\alpha : F \to G$ is a structure with the following properties:
(a) For each open subset $U \subseteq X$, there is a function $\alpha(U) : F(U) \to G(U)$.
(b) For all open subsets $U, V \subseteq X$ with $V \subseteq U$, the diagram below commutes [1, 006E].

$$\begin{array}{ccc} F(U) & \xrightarrow{\alpha(U)} & G(U) \\ \downarrow{\scriptstyle \rho_{UV}} & & \downarrow{\scriptstyle \rho'_{UV}} \\ F(V) & \xrightarrow{\alpha(V)} & G(V) \end{array}$$

```
structure morphism (F G : presheaf X) :=
(map      : 8 (U), F U  !  G U)
(commutes : 8 (U V) (HVU : V    U),
  (G.res U V HVU)   (map U) = (map V)   (F.res U V HVU))
```

If G is F and $(U) = id_{F(U)}$, we call the *identity morphism* and denote it id_F. A morphism $: F \; ! \; G$ is an *isomorphism* if there exists a morphism $: G \; ! \; F$ such that $= id_F$ and $= id_G$, where the composition of morphisms is de ned in the obvious way.

```
structure iso (F G : presheaf X) :=
(mor : F   !  G)
(inv : G   !  F)
(mor_inv_id : mor  } inv = id F)
(inv_mor_id : inv  } mor = id G)
```

Note that we have introduced some notation: `!` stands for `morphism` as de ned above and `}` is a shortcut for composition of morphisms. Lean allows the user to introduce new notation, which is often useful to make the code more readable.

The de nition for isomorphism above is perfectly ne to write statements of the form `given an explicit isomorphism between $F$ and $G$ then $:::$'. However, in mathematics we tend to work non-constructively so we would like to say `if there is an isomorphism between F and G then $:::$' instead. Therefore we de ne `F = G` to mean `nonempty (iso F G)`, which is a way of saying that `iso F G` is a type such that an element can be chosen using the axiom of choice.

Let us now give a couple of examples of presheaves.

Example 27. Let X be a topological space and S a set, then the constant presheaf given by $F(U) = S$ and with restriction maps de ned to be id_S is trivially a presheaf.

Example 28. Consider the real line R and let F be the presheaf de ned by the rule $F(U) = ff : U \; ! \; R \; j \; f$ is continuous and boundedg for open U R. Restriction maps are simply restrictions on the domains of the functions hence it is clear that it is a presheaf.

In this last example we observe that the sections have more structure than just elements of sets. In fact, each $F(U)$ is naturally a ring, given by the operations $(f + g)(x) = f(x) + g(x)$ and $fg(x) = f(x)g(x)$. We will mostly be concerned with presheaves of this kind.

Definition 29. A presheaf F on a topological space X is called a *presheaf of rings* if for each open set U X, $F(U)$ is a ring and all the restriction maps are ring homomorphisms [1, 006N].

```
structure presheaf_of_rings (X : Type u) [topological_space X]
extends presheaf X :=
(Fring           : 8 (U), comm_ring (F U))
(res_is_ring_hom : 8 (U V) (HVU : V    U), is_ring_hom (res U V HVU))
```

Obviously, the de nition of morphism needs to be extended.

Definition 30. A *morphism of presheaves of rings* $: F \; ! \; G$ on a topological space X is a morphism of the underlying presheaves of sets such that (U) is a ring homomorphism for every open $U \quad X$ [1, 006N].

```
structure morphism (F G : presheaf_of_rings X)
extends presheaf.morphism F.to_presheaf G.to_presheaf :=
(ring_homs : 8 (U), is_ring_hom (map U))
```

The identity map is a ring homomorphism and composition preserves ring homomorphisms so isomorphisms of presheaves of rings are de ned in the same way as isomorphisms of presheaves of sets.

3.2.2 Sheaves

In this subsection we present the sheaf condition and explain why it is crucial to do geometry. As before, we also de ne maps between sheaves and sheaves of rings.

Definition 31. A presheaf F on a topological space X is a sheaf if for any oen $U \quad X$ and any fU_ig covering U the following two conditions hold:
(a) If $s;t \; 2 \; F(U)$ is such that $sj_{U_i} = tj_{U_i}$ for all i, then $s = t$.
(b) Given $s_i \; 2 \; F(U_i)$ for each i, if $s_i j_{U_i \setminus U_j} = s_j j_{U_i \setminus U_j}$ for all i and j then there exists $s \; 2 \; F(U)$ such that $sj_{U_i} = s_i$ for all i [1, 006T].

```
variables (X : Type u) [T : topological_space X]

def locality (F : presheaf X) :=
8 {U} (OC : covering U) (s t : F U),
(8 i, F.res U (OC.Uis i) (subset_covering i) s =
      F.res U (OC.Uis i) (subset_covering i) t)  !
s = t

def gluing (F : presheaf X) :=
8 {U} (OC : covering U),
8 (s :   i, F (OC.Uis i)),
(8 j k, res_to_inter_left F (OC.Uis j) (OC.Uis k) (s j) =
        res_to_inter_right F (OC.Uis j) (OC.Uis k) (s k))  !
9 S, 8 i, F.res U (OC.Uis i) (subset_covering i) S = s i

structure sheaf extends presheaf X :=
(locality : locality to_presheaf)
(gluing   : gluing to_presheaf)
```

Informally, the gluing condition says that if a set of `functions' $\{f_i\}$ perhaps de ned on di erent intervals coincide in their intersections then there is a `function' f that contains all of them. One can think of continuous piecewise functions for instance as a simple example of this phenomenon. The locality condition ensures that f is unique.

Dealing with sheaves formally turns out to be way more delicate than that. In fact, we will spend all of section 3.4 essentially proving the sheaf condition for a special kind of presheaf. In order to give a deeper insight into it, let us look back at the examples from the previous subsection.

Example 32. The constant presheaf de ned in Example 27 is a sheaf if and only if S is a singleton. The sheaf condition enforces $F(?)$ to be a nal object, that is, an object in a category such that every other object maps to it uniquely.

Example 33. The presheaf of bounded real-valued continuous functions from Example 28 is not a sheaf. Let $U_i = (i - 1; i + 1)$ for each $i \in \mathbb{Z}$. Clearly $\bigcup_i U_i = \mathbb{R}$. Now consider the functions $f_i : U_i \to \mathbb{R}$ given by $f_i(x) = x$. Each f_i is certainly continuous and $|f_i(x)| \leq i + 1$ so $f_i \in F(U_i)$. Suppose for a contradiction that some $f \in F(\mathbb{R})$ satis ed the gluing condition. By assumption $|f(x)| \leq M$ for some M but $|f_{M+1}(x)| \not\leq M$.

Example 34. However, if we drop the boundedness condition and de ne, for open $U \subseteq \mathbb{R}$, $F(U) = \{f : U \to \mathbb{R} \mid f \text{ continuous}\}$ with the obvious restriction maps, we obtain a sheaf. It is a standard application of the pasting lemma from topology.

As a nal example and to show the sort of objects on which sheaves are actually useful, we generalize the idea of the previous one.

Example 35. Let X be a smooth manifold (a topological space where we can locally do calculus) then for any open $U \subseteq X$ the rule $F(U) = C^1_{X;\mathbb{R}}(U) = \{f : U \to \mathbb{R} \mid f \text{ smooth}\}$ gives rise to a sheaf. In fact, he same idea works for a complex analytic manifold.

Note that all of the examples that we have seen so far come from analysis, where they appear very naturally. As it was mentioned in section 1.2 one of the purposes of a scheme is precisely to transport these ideas from analysis to algebra.

Again, the $F(U)$ often happen to be rings so we de ne `sheaf_of_rings` in the same way but extending `presheaf_of_rings` instead.

Finally, *morphisms of sheaves* are simply morphisms of the underlying presheaves.

3.2.3 Stalks

We have seen how presheaves and sheaves let us argue about what happens locally looking into subsets and their intersections. However, it is not clear at all what happens at a single element. Stalks allow us to focus on a single element by considering all the open sets it belongs to at once.

Formally we say that the *stalk* of a presheaf F at $x \in X$ is the colimit of the sets $F(U)$ for open $U \ni x$ and denote it F_x. This means that there are maps $F(U) \to F_x$ for each open $U \ni x$ such that all diagrams commute and for any other object S satisfying this property there is a unique map $F_x \to S$. What this means is that F_x contains the information common

to all the $F(U)$ with $x \in U$ and nothing else. Intuitively, we see how this information must have something to do with the behaviour of the presheaf around x.

We can explicitly de ne this colimit.

Definition 36. Let X be a topological space and F a presheaf of sets on X. The *stalk* of F at a point $x \in X$ is the set of all pairs $(U; s)$ where $x \in U$ and $s \in F(U)$ under the equivalence relation given by $(U; s) \sim (V; t)$ if an only if there exists an open neighbourhood of x such that $W \subseteq U \cap V$ and $s|_W = t|_W$ [1, 0078].

```
structure stalk.elem :=
(U   : opens X)
(HxU : x ∈ U)
(s   : F U)

def stalk.relation : stalk.elem F x → stalk.elem F x → Prop :=  λ Us Vt,
    ∃ (W : opens X) (HxW : x ∈ W) (HWU : W ≤ Us.U) (HWV : W ≤ Vt.U),
    F.res Us.U W HWU Us.s = F.res Vt.U W HWV Vt.s

instance stalk.setoid : setoid (stalk.elem F x) :=
{ r     := stalk.relation F x,
  iseqv := stalk.relation.equivalence F x }

def stalk := quotient (stalk.setoid F x)
```

First we check that stalks are well-de ned and that they satisfy the required property.

Proposition 37. The relation $\sim$ as above is indeed an equivalence relation [1, 0078].

```
lemma stalk.relation.equivalence : equivalence (stalk.relation F x)
```

Proof. Re exivity and symmetry are obvious. For transitivity suppose $(U; a) \sim (V; b)$ and $(V; b) \sim (W; c)$. This implies that there exists $R \subseteq U \cap V$ and $S \subseteq V \cap W$ such that $a|_R = b|_R$ and $b|_S = c|_S$ so simply choose $R \cap S \subseteq U \cap W$. Then, using the composition property of restriction maps, $a|_{R\cap S} = (a|_R)|_{R\cap S} = (b|_R)|_{R\cap S} = b|_{R\cap S} = (b|_S)|_{R\cap S} = (c|_S)|_{R\cap S} = c|_{R\cap S}$ and we are done. □

Proposition 38. The stalk F_x de ned as above is indeed $colim_{x\in U} F(U)$ [1, 0078].

```
def to_stalk (U : opens X) (HxU : x ∈ U) : F U → stalk F x :=
  s, ⟦{U := U, HxU := HxU, s := s}⟧

variables (S : Type w) [decidable_eq S]
variables (G :  Π U, x ∈ U → F U → S)
```

```
variables (HG : ∀ V U (HVU : V ⊆ U) (HxV : x ∈ V),
    (G V HxV) ≫ (F.res U V HVU) = G U (HVU HxV))

def to_stalk.rec : stalk F x → S :=
  λ y, quotient.lift_on' y (λ Us, G Us.1 Us.2 Us.3)
(λ hU, HxU, si hV, HxV, ti hW, HxW, HWU, HWV, Hresi,
by dsimp; erw [← HG W U HWU s HxW, ← HG W V HWV t HxW, Hres])

lemma to_stalk.rec_to_stalk (U : opens X) (HxU : x ∈ U)
: (to_stalk.rec F x S G HG) ≫ (to_stalk F x U HxU) = G U HxU
```

Proof. Obvious. □

What just happened? Well, for Lean this lemma is completely obvious and the proof is just `rfl` (or `by refl`). It remains to explain these definitions and justify that this is actually the result that we need, which might not be so clear at first glance. The function `to_stalk` is the map $F(U) \to F_x$ sending $s \mapsto (U; s)$, whose existence is the first requirement, and we found it! Now suppose that there is some S such that there are maps $g_U : F(U) \to S$ (`G`) and they commute with the restriction maps (`HG`). Firstly, we need to come up with a map $F_x \to S$. That is exactly `to_stalk.rec` which, less obviously from the definition, is given by $(U; s) \mapsto g_U(s)$. The reason why the definition includes what looks like a proof is because one needs to convince Lean that this is actually well-defined. In particular, we show that if $(U; s) \sim (V; t)$ then $g_U(s) = g_V(t)$ so we look at the $W \subseteq U \cap V$ where $s|_W = t|_W$. Because of compatibility with restrictions, we see that $g_U(s) = g_W(s|_W) = g_W(t|_W) = g_V(t)$ so the map is well-defined. However, after all this convincing, it is straightforward that $s \mapsto (U; s) \mapsto g_U(s)$ is the same as $s \mapsto g_U(s)$ for all $U \ni x$. Uniqueness is also clear now as any map $F_x \to S$ has to send $(U; s)$ to $g_U(s)$ to satisfy the lemma or, in other words, it needs to be exactly `to_stalk.rec`.

Again, we mostly care about the case when F is a presheaf of rings.

Definition 39. Given a presheaf of rings F on a topological space X, the *stalk of rings* of F at $x \in X$ is the stalk of the underlying presheaf of sets of F at x [1, 007G].

```
definition stalk_of_rings := stalk F.to_presheaf x
```

There are a couple of checks that are necessary before we move forward.

Proposition 40. The stalk of rings F_x is a ring [1, 007G].

```
instance stalk_of_rings_is_comm_ring : comm_ring (stalk_of_rings F x)
```

Proof. We let $(X; 0) \in F_x$ be the zero element and $(X; 1) \in F_x$ be the unity. Addition is given by $(U; s) + (V; t) = (U \cap V; s|_{U\cap V} + t|_{U\cap V})$ and multiplication by $(U; s) \cdot (V; t) =$

$(U \setminus V; sj_{U \setminus V}\ \ tj_{U \setminus V})$. One checks in a similar fashion as before that these are well-de ned by sending everything to the subset where the elements coincide. All the ring axioms need to be checked but in every case it boils down to nding a subset W small enough where everything coincides. The axioms are then be inherited from the ring structure of $F(W)$. □

Proposition 41. The stalk of rings F_x corresponds to $colim_{x2U} F(U)$ in the category of rings [1, 007G].

```
variables [comm_ring S] [Hg_ring_hom :   U, is_ring_hom (G U)]

lemma to_stalk.is_ring_hom (U : opens X) (HxU : x 2 U)
: is_ring_hom (to_stalk F x U HxU)

lemma to_stalk.rec_is_ring_hom : is_ring_hom (to_stalk.rec F x S G Hg)
```

Proof. We just need to check that the maps in Proposition 38 are ring homomorphisms. The rst lemma follows fairly immediately from the fact that the restriction maps are ring homomorphisms. For the second one, we use the fact that g_U's are ring homomorphisms and that they are compatible with restriction maps. □

3.2.4 Bases

The last bit of machinery that needs to be developed has to do with de ning a presheaf on a basis and extending it to the whole space. We de ne presheaves, sheaves and stalks on a basis and then specialise these de nitions to the case of rings.

Definition 42. Let X be a topological space and B a basis for X, a *presheaf of sets on a basis* F is a presheaf of sets on X just de ned on the $U\ 2\ B$ [1, 009I].

```
structure presheaf_on_basis (X : Type u) [T : topological_space X]
{B : set (opens X)} (HB : opens.is_basis B) :=
(F     :   {U}, U 2 B  !  Type v)
(res   : 8 {U V} (BU : U 2 B) (BV : V 2 B) (HVU : V   U), F BU  !  F BV)
(Hid   : 8 {U} (BU : U 2 B), (res BU BU (set.subset.refl U)) = id)
(Hcomp : 8 {U V W}
  (BU : U 2 B) (BV : V 2 B) (BW : W 2 B) (HWV : W   V) (HVU : V   U),
  res BU BW (set.subset.trans HWV HVU) =
  (res BV BW HWV)   (res BU BV HVU))
```

Morphisms are de ned in an analogous way as for presheaves on the whole space.

Next we de ne the sheaf condition for presheaves on basis. We need to be a little bit more careful here. Previously, we were given an open cover fU_ig for some $U\ \ X$ and considered intersections of the U_i's. The issue is that if $U_i; U_j\ 2\ B$ it is not necessarily the case that

$U_i \setminus U_j \in B$. However, by basic topology, we know that there exist $\{U_{ijk}\}$ covering $U_i \setminus U_j$ such that $U_{ijk} \in B$. The following structure carries this information.

```
structure covering_basis (U : opens X) extends covering U :=
{Iij       :   !    ! Type v}
(Uijks     :   (i j), Iij i j  ! opens X)
(BUis      : ∀ i, Uis i ∈ B)
(BUijks    : ∀ i j k, Uijks i j k ∈ B)
(Hintercov : ∀ i j, ⋃ (Uijks i j) = Uis i \ Uis j)
```

Now, instead of considering the restriction to $U_i \setminus U_j$, we can consider the restrictions to U_{ijk} for all k.

Definition 43. Let F be a presheaf of sets on a topological space X and let B be a basis of X. We say that F is a *sheaf of sets on the basis B* if it satisfies the following condition. Pick any open $U \subseteq X$ and $\{U_i\}$ covering U. If $s_i \in F(U_i)$ satisfy $s_i|_{U_{ijk}} = s_j|_{U_{ijk}}$ for all k where $\{U_{ijk}\} \subseteq B$ is any open cover for $U_i \setminus U_j$ then there is a unique $s \in F(U)$ such that $s|_{U_i} = s_i$ for all i [1, 009J].

```
def is_sheaf_on_basis (F : presheaf_on_basis X HB) : Prop :=
∀ {U} (BU : U ∈ B) (OC : covering_basis U),
∀ (s :   i, F (OC.BUis i)),
(∀ i j k, F.res (OC.BUis i) (OC.BUijks i j k)
            (subset_covering_basis_inter_left i j k) (s i) =
          F.res (OC.BUis j) (OC.BUijks i j k)
            (subset_covering_basis_inter_right i j k) (s j))  !
∃! S, ∀ i, F.res BU (OC.BUis i) (subset_covering i) S = s i
```

Note that before we defined a sheaf as a structure extending a presheaf which, in addition, satisfied the sheaf condition. Here, instead, the sheaf condition is defined as a proposition. We have both versions in both cases and, mathematically, there is no difference between one and the other.

It remains to consider stalks on a basis.

Definition 44. Suppose B is a basis for X, F is a presheaf on B and $x \in X$. We define F_x, the *stalk on the basis B*, as the set of pairs $(U; s)$ where $x \in U \in B$ under the equivalence relation $(U; s) \sim (V; t)$ if and only if there exists $W \subseteq U \setminus V$ such that $x \in W$, $W \in B$ and $s|_W = t|_W$ [1, 009H].

```
structure stalk_on_basis.elem :=
(U  : opens X)
(BU : U ∈ B)
(Hx : x ∈ U)
(s  : F BU)
```

The proof that is an equivalence relation is analogous to the one in Proposition 37. Moreover, it is also true that $F_x = colim_{x2U;U2B} F(U)$ arguing as in Proposition 38.

Again, in exactly the same way as before, we extend these de nitions to the case of rings.

Definition 45. A presheaf F on the basis B is a *presheaf of rings on the basis B* if for every U 2 B we have that $F(U)$ is a ring and every restriction map $_{UV} : F(U) ! F(V)$ is a ring homomorphism [1, 009P].

```
structure presheaf_of_rings_on_basis (X : Type u) [topological_space X]
{B : set (opens  )} (HB : opens.is_basis B)
extends presheaf_on_basis   HB :=
(Fring           : 8 {U} (BU : U 2 B), comm_ring (F BU))
(res_is_ring_hom : 8 {U V} (BU : U 2 B) (BV : V 2 B) (HVU : V   U),
  is_ring_hom (res BU BV HVU))
```

The de nitions for a sheaf of rings on a basis and a stalk of rings on a basis are the same as the de nitions for a sheaf on a basis and a stalk on a basis respectively with the assumption that the underlying presheaf is a presheaf of rings on a basis.

Before we pointed out that if B is a basis and $U;V$ 2 B we cannot assume that $U \setminus V$ 2 B. However, if a basis satis es this condition we say that it is a *standard basis*. It makes sense to consider standard bases. In fact, this idea was introduced previously in Lemma 14 where we proved that the standard opens $fD(f)g$ form a standard basis for $Spec(R)$.

Hereafter we assume that any basis we encounter is standard. All of the results hold for a general basis but that adds some uninteresting complexity to the proofs, which are already fairly technical.

Lemma 46. If F is a presheaf of rings on a standard basis then for all x 2 X we have that F_x is a ring [1, 009P].

```
instance comm_ring
: comm_ring (stalk_of_rings_on_standard_basis Bstd F x)
```

Proof. Under the assumption that the basis is standard the proof is exactly the same as that of Proposition 40. □

The nal result in this subsection relates the sheaf condition on a basis with compactness of the basis elements in the sense of Lemma 22.

Lemma 47. Suppose we are given a presheaf F on a basis B which is standard and every cover of a basis element admits a nite subcover. If F satis es the sheaf condition for nite open covers then F satisi es the sheaf condition for arbitrary covers [1, 009K].

```
def is_sheaf_on_standard_basis_cofinal_systems : Prop :=
∀ {U} (BU : U ∈ B) (OC : covering_standard_basis B U) (H : fintype OC. ),
locality Bstd F BU OC ∧ gluing Bstd F BU OC

lemma is_sheaf_on_standard_basis_of_cofinal
(F : presheaf_on_basis X HB)
(Hcompact : basis_elements_finite_subcover Bstd)
: is_sheaf_on_standard_basis_cofinal_systems Bstd F ↔
  is_sheaf_on_standard_basis Bstd F
```

Proof. Pick any $U \in B$ and any open cover $\{U_i\}$ of U. By compactness of the basis, we can find, after possibly renaming, $U = \{U_1, \ldots, U_n\}$ covering U and, by assumption, F satisfies the sheaf condition on U.

For locality suppose that $s, t \in F(U)$ and $s|_{U_i} = t|_{U_i}$ for all i then, in particular, $s|_{U_j} = t|_{U_j}$ for $1 \leq j \leq n$ so $s = t$ and we are done.

To prove the gluing condition, suppose that we have $s_i \in F(U_i)$ such that $s_i|_{U_i \cap U_j} = s_j|_{U_i \cap U_j}$ for all i and j. There exists $s \in F(U)$ such that $s|_{U_j} = s_j$ for $1 \leq j \leq n$. We claim that $s|_{U_i} = s_i$ for all i. Fix an i and define $V_j = U_i \setminus U_j$ for $1 \leq j \leq n$. Since B is standard, we have that $V_j \in B$. Moreover $V = \{V_1, \ldots, V_n\}$ covers U_i. Consider $t_j = s|_{V_j}$. Then for every $1 \leq k, l \leq n$ we have $t_k|_{V_k \cap V_l} = s|_{V_k \cap V_l} = t_l|_{V_k \cap V_l}$ so by the gluing condition on finite covers applied to U_i and V we have that there exists $t \in F(U_i)$ such that $t|_{V_k} = t_k$. Thus $t|_{V_k} = s|_{V_k} = (s|_{U_i})|_{V_k}$ so, by locality, $t = s|_{U_i}$. But also $t|_{V_k} = s|_{V_k} = (s|_{U_k})|_{V_k} = s_k|_{V_k} = s_i|_{V_k}$ and therefore $t = s_i$, again by locality. We conclude that $s|_{U_i} = s_i$ as required. □

In [1, 009K] this lemma is proved in more generality. The basis does not need to be standard or compact. Consider for any U the set of all coverings $Cov_B(U)$ preordered by refinement. A subset $C(U) \subseteq Cov_B(U)$ is called a *cofinal system* if for every $U \in Cov_B(U)$ there is some $V \in C(U)$ such that $V < U$. They prove that if for every $U \in B$ there is some cofinal system $C(U)$ such that for all $U \in C(U)$ the sheaf condition holds for U then it holds on B. In our case, since we assume that the basis is compact we take as our cofinal system the set of all finite covers.

3.2.5 Extensions

In the previous subsection we saw how one can do sheaf theory on the basis. In a philosophical sense, the basis has all the information of the topological space so we would expect that a presheaf F defined on a basis B could be naturally extended to a presheaf F^{ext} on the whole space. Moreover, given $U \in B$ we would like $F(U) = F^{ext}(U)$. It turns out that with our construction of F^{ext} this property is not necessarily true unless F is a sheaf. We present said construction and study how it relates to the given presheaf on a basis.

Definition 48. Suppose F is a presheaf on a basis B. Given any open $U \subseteq X$ not necessarily in the basis we define the *presheaf extension* as follows [1, 009N].

$$F^{ext}(U) = \{((U_x, s_x)) \in \prod_{x \in U} F_x \mid \exists V \in B \text{ such that } x \in V \subseteq U \text{ and } \exists t \in F(V) \text{ with } (U_y, s_y) = (V, t) \text{ for all } y \in U \setminus V\}$$

```
def presheaf_extension (F : presheaf_on_basis X HB) : presheaf X :=
{ F := λ U, {s : Π (x ∈ U), stalk_on_basis F x //
    ∀ (x ∈ U), ∃ (V) (BV : V ∈ B) (Hx : x ∈ V) (t : F BV),
    ∀ (y ∈ U \ V), s y = ⟨_, ⟦{U := V, BU := BV, Hx := H.2, s := t}⟧⟩},
  res := λ U W HWU FU,
        { val := λ x HxW, (FU.val x (HWU HxW)),
          property := λ x HxW,
            begin
              rcases (FU.2 x (HWU HxW)) with ⟨V, ⟨BV, ⟨HxV, h⟩⟩⟩, HFV⟩⟩⟩⟩,
              use [V, BV, HxV, λ _ _ _ _ h, h],
              rintros y ⟨HyW, HyV⟩,
              rw (HFV y ⟨HWU HyW, HyV⟩),
            end },
  Hid := λ U, funext (λ x, subtype.eq rfl),
  Hcomp := λ U V W HWV HVU, funext (λ x, subtype.eq rfl)}
```

It is worth explaining the work that needed to be done to define `res`. Suppose that $W \subseteq U$, then the restriciton maps $\rho_{UW} : F^{ext}(U) \to F^{ext}(W)$ arise from the obvious restriction $\prod_{x \in U} F_x \to \prod_{x \in W} F_x$. For any $x \in W$ we have $x \in U$ so we choose the $V \in B$ and $t \in F(V)$ given by the property then certainly $(U_x, s_x) = (V, t)$ for all $y \in W \setminus V \subseteq U \setminus V$. This is exactly the proof that appears in the definition.

We can also require that F is a presheaf of rings on a basis, in which case the resulting object is a presheaf of rings. Proving that $F^{ext}(U)$ is a ring is a consequence of F_x being a ring and products of rings being rings. The restriction maps are easily seen to be ring homomorphisms if we think of them as restrictions from a product of rings.

We now prove an important property about this construction.

Lemma 49. With the same definition as above, F^{ext} is a sheaf [1, 009N].

```
lemma extension_is_sheaf (F : presheaf_on_basis X HB)
: is_sheaf (presheaf_extension F)
```

Proof. Fix an open set $U \subseteq X$ and a covering $\{U_i\}$ of U.

First, we prove locality. Let $s, t \in F^{ext}(U)$ be such that $s|_{U_i} = t|_{U_i}$ for all i. In particular $s_x = t_x$ in F_x for $x \in U_i$. What we need to show is that for any $x \in U$, $s_x = t_x$. But $U = \bigcup U_i$ so any $x \in U$ is in $x \in U_i$ for some i and the result follows.

Asserting the gluing property will require some more work. Pick any $s_i \in F^{ext}(U_i)$ such that $s_i|_{U_i \setminus U_j} = s_j|_{U_i \setminus U_j}$ for all i and j. We define the global section $s \in F(U)$, prove that it is well-defined and then show that it satisfies the required property. For $x \in U_i$ write $s_i(x)$ for $s_i \in F_x$, i.e. value of s_i in the stalk F_x. Let s be given by $(s_i(x))_{x \in U}$ where the i is chosen such that U_i contains x. Note that if $x \in U_i \setminus U_j$ then $s_i(x) = s_j(x)$ by the equivalence relation defined on stalks. To prove the property, pick any $x \in U$ then $x \in U_i$ for some i and there is some $x \in V \in B$ and $\in F(V)$ such that $s_i(x) = (V; \ )$ and therefore the property holds for every element of U so $s \in F(U)$ as required. By construciton, $s|_{U_i} = (s_i(x))_{x \in U_i} = s_i$ and so the sheaf condition is proved. □

The exact same proof holds if we start with a presheaf of rings instead. As a matter of fact, from now on we will only consider presheaves of rings making the next section the natural continuation of this one. For the same reason explained in Lemma 47, we will assume that the bases are standard. It simplifies the calculations and we know that it is enough to give a complete definition of a scheme.

At this point there are several questions that come to mind. It is slightly suspicious that a sheaf appeared from nowhere. However, it makes some sense because we are setting up the $F^{ext}(U)$'s in a way that they can be glued with the $F(U)$ for $U \in B$ which is the only information we have at first. This construction forces that if sections coincide in intersections in the sense of the sheaf condition then they can be glued together. A way to think about it is that we start with $U \subseteq X$ and for each $x \in U$ we gather all the information we can about the `functions' on it, which is in $F_x$ defined on the basis. Then we can choose a small enough neighbourhood of $x$ and a `function' on it in a way that it respects the existing `functions' on the basis elements. It is related to the more general idea of sheafification which we do not cover but that can be found in [1, 007X].

The argument above will be formalised at the end of this subsection when we explicitly use the sheaf condition to prove that there is an isomorphism $F^{ext}(U) = F(U)$. Before we tackle such issue, we study the extension locally. It turns out that it behaves quite nicely.

Lemma 50. Let F be a presheaf on a standard basis, then for all $x \in X$ we have $F^{ext}_x = F_x$ [1, 009H].

```
lemma to_stalk_extension.equiv
(F : presheaf_of_rings_on_basis X HB) {U : opens X} (BU : U ∈ B)
: stalk_of_rings_on_standard_basis Bstd F x ≃r
stalk_of_rings (presheaf_of_rings_extension Bstd F) x
```

Proof. For $U \in B$, let $\varphi_U : F(U) \to F^{ext}(U)$ be the canonical map $\varphi_U(s) = (s) \in \bigcup_{x \in U} F_x$. Note that it trivially satisfies the property of Definition 48. We need to come up with a map $\psi : F_x \to F^{ext}_x$. Define $\psi((U;s)) = (U; \varphi_U(s)) \in F^{ext}_x$. We show that this map is injective, surjective and a ring homomorphism. Supose that $(U; \varphi_U(s)) = (V; \varphi_V(t))$ then there exists $W \subseteq U \setminus V$ such that $\varphi_U(s)|_W = \varphi_V(t)|_W$. This means that $s = t$ on W so $(U;s) = (V;t)$ hence ψ is injective. Let $(V;t) \in F^{ext}_x$. By definition, t has the property that there exists some $U \in B$ such that $x \in U$ and $s \in F(U)$ with $(U;s) = t_y$ for all $y \in U \setminus V \in B$, which in our notation translates to $(U \setminus V; \varphi_{U \setminus V}(s)) = (V;t)$, so the map is surjective. The fact that

it is a ring homomorphism comes from the ring structure de ned on stalks of presheaves of rings proved in Lemma 46. □

We nish this section with the main result about extensions of presheaves.

Lemma 51. If F is a sheaf of rings on a standard basis B then $F^{ext}(U) = F(U)$ for $U \in B$ [1, 009N].

```
lemma to_presheaf_of_rings_extension.ring_equiv
(F : presheaf_of_rings_on_basis X HB)
(HF : is_sheaf_on_standard_basis Bstd F.to_presheaf_on_basis)
{U : opens X} (BU : U ∈ B)
: F BU ≃r (presheaf_of_rings_extension F Bstd) U
```

Proof. We prove that the map φ_U de ned in the previous lemma is an isomorphism of rings for every $U \in B$.

Suppose that $\varphi_U(s) = \varphi_U(t)$ for some $s, t \in F(U)$. This means that $(U, s) = (U, t)$ in F_x for every $x \in U$. We deduce that there exist open $W_x \subseteq U$ with $x \in W_x$ such that $s|_{W_x} = t|_{W_x}$ for all x. Clearly $\{W_x\}$ is an open cover of U so, by locality, we deduce that $s = t$ and hence φ_U is injective.

Now pick any $(s_x) \in F^{ext}(U)$. By de nition, for every $x \in U$ there is a basis element $V_x \in B$ with $x \in V_x$ and $\sigma_x \in F(V_x)$ such that for all $y \in U \cap V_x = W_x \in B$ (because B is standard) we have that $s_y = (V_x, \sigma_x)$. Similarly as before, we see that $\{W_x\}$ covers U. We claim that $\sigma_x|_{W_x \cap W_y} = \sigma_y|_{W_x \cap W_y}$ for all $x, y \in U$. Note that for all $z \in W_x \cap W_y$ we have that $(V_x, \sigma_x) = s_z = (V_y, \sigma_y)$ so we can nd W_{xyz} such that $z \in W_{xyz} \in B$ and $\sigma_x|_{W_{xyz}} = \sigma_y|_{W_{xyz}}$. We can therefore cover $W_x \cap W_y$ with $\{W_{xyz}\}$ and apply locality to this intersection to deduce the claim. By the gluing condition and the claim we nd a section $s \in F(U)$ such that $s|_{W_x} = \sigma_x$ for all $x \in U$ and then $\varphi_U(s) = ((U, s)) = ((W_x, \sigma_x)) = ((V_x, \sigma_x)) = (s_x)$, hence we have proved surjectivity.

By the standard argument, φ_U is a ring homomorphism. □

3.3 Locally ringed spaces

Finally, we have enough machinery to start thinking about schemes. As we will see, a scheme is a locally ringed space with a certain property hence we rst de ne locally ringed spaces following [1, 01HA].

A *ringed space* is simply a topological space X together with a sheaf of rings $\mathcal{O}_X$. Note that we use $\mathcal{O}$ for ringed spaces instead of F, following the standard convention. When we study algebraic geometry we often want to look in one sense or another at the ring of `functions' that characterise a single point $x \in X$. These are often called *germs* and live in the stalk of $\mathcal{O}_X$ at x. We might expect for all the germs which vanish at x to form an ideal as in general we want that if $f(x) = 0$ and $g(x) = 0$ then $(f + g)(x) = 0$. This would be obvious in the general setting but sheaves allow more exibility. The canonical examples such as

the sheaf of real-valued continuous functions on a topological space X satisfy this property. Moreover, by basic commutative algebra, if the nonunits form an ideal, it must be maximal. If a function does not vanish at x then we can find a small neighbourhood where it is not zero anywhere so it is invertible as an element of the stalk. This observation justifies the following definition.

Definition 52. A *locally ringed space* is a topological space X with an associated sheaf of rings $\mathcal{O}_X$ such that at every $x \in X$ the stalk $\mathcal{O}_{X,x}$ is a local ring [1, 01HB].

```
structure locally_ringed_space (X : Type u) [topological_space X] :=
(O       : sheaf_of_rings X)
(Hstalks : ∀ x, is_local_ring (stalk_of_rings O.F x))
```

Introducing the notion of a locally ringed space has been one of the main contributions of this project. The definition of a scheme defined in the first attempt did not define a scheme as a locally ringed space. One does not need to impose the condition on stalks as it is true by construction and that is why we said that the two definitions were mathematically equivalent. Nevertheless, when doing formal mathematics it is in general a good practice to follow the mathematical definition as much as follows as there is no obvious way of checking that the definition itself is the correct one.

If we want to treat schemes as a special kind of locally ringed spaces, it is useful to study the maps between them. They will be given by the maps between their underlying presheaves. However, morphisms of presheaves are of no help here as, in general, we want to define maps between two locally ringed spaces on different topological spaces.

Assume that X and Y are topological spaces and $f : X \to Y$ a continuous map.

Definition 53. Given a presheaf $\mathcal{F}$ on X, the *pushforward* $f_*\mathcal{F}$ is a presheaf on Y defined by the rule $f_*\mathcal{F}(U) = \mathcal{F}(f^{-1}(U))$ [1, 008C].

```
def pushforward (F : presheaf X) : presheaf Y :=
{ F :=   U, F (opens.comap Hf U),
  res :=   U V HVU, F.res
    (opens.comap Hf V) (opens.comap_mono Hf V U HVU),
  Hid :=   U, F.Hid (opens.comap Hf U),
  Hcomp :=   U V W HWV HVU,
    F.Hcomp (opens.comap Hf U) (opens.comap Hf V) (opens.comap Hf W)
            (opens.comap_mono Hf W V HWV) (opens.comap_mono Hf V U HVU) }
```

Pushforwards are the easiest maps to define as they follow naturally from continuity. Now, we would like to define a presheaf on X from a presheaf on Y using f. This can be done in general but we will only worry about a particular type of maps, which will make the definition much more straightforward. In particular we will assume that if $U \subseteq X$ is open then $f(U) \subseteq Y$ is open. There is only one place where we need this definition, namely to

de ne a scheme, and the maps there could, in principle, just be inclusions $\ : U \ ! \ X$ to some subset $U \quad X$, therefore the following construction is general enough.

Definition 54. If f is an open map then the *open pullback* $f \ F$ is given by the rule $f \ F(U) = F(f(U))$ [1, 008F].

```
variable (Hf' : 8 (U : opens X), is_open (f '' U))

def open_pullback (F : presheaf Y) : presheaf X :=
{ F :=   U, F (opens.map Hf' U),
  res :=   U V HVU, F.res
    (opens.map Hf' U) (opens.map Hf' V) (opens.map_mono Hf' V U HVU),
  Hid :=   U, F.Hid (opens.map Hf' U),
  Hcomp :=   U V W HWV HVU,
    F.Hcomp (opens.map Hf' U) (opens.map Hf' V) (opens.map Hf' W)
            (opens.map_mono Hf' W V HWV) (opens.map_mono Hf' V U HVU)}
```

For convenience, we package the data needed to de ne a pullback under an *open immersion*, which is a continuous injective open map which is, in essence, an inclusion map.

```
structure open_immersion_pullback (F : presheaf Y) :=
(f       : X ! Y)
(Hf₁     : continuous f)
(Hf₂     : 8 (U : opens X), is_open (f '' U))
(Hf₃     : function.injective f)
(range   : opens Y := hf '' set.univ, Hf₂ opens.univi)
(carrier : presheaf X := open_pullback Hf₂ F)
```

Instead of generating new presheaves, in some situations it is useful to start with F a presheaf on X and G a presheaf on Y and see whether f can map from one to the other respecting restrictions.

Definition 55. An *f-map* from G to F is a set of maps $\ _U : G(U) \ ! \ F(f \ ^1(U))$ such that the following diagram commutes.

$$\begin{array}{ccc} G(U) & \xrightarrow{\ U\ } & F(f\ ^1(U)) \\ \downarrow{\scriptstyle \ ^0_{UV}} & & \downarrow{\scriptstyle \ _{UV}} \\ G(V) & \xrightarrow{\ V\ } & F(f\ ^1(V)) \end{array}$$

We denote them $f^{\#} : G \ ! \ F$ [1, 008J].

```
structure fmap (F : presheaf   ) (G : presheaf   ) :=
(map      : 8 (U), G U  !  F (opens.comap Hf U))
(commutes : 8 (U V) (HVU : V    U),
  (map V)   (G.res U V HVU)
= (F.res (opens.comap Hf U) (opens.comap Hf V)
  (opens.comap_mono Hf V U HVU))   (map U))
```

These maps can be composed in the obvious way. Moreover, an f-map between G and F gives for every x 2 X an induced map $f_x^{\#} : G_{f(x)}$! F_x given by $(U;s)$ 7! $(f^{-1}(U);\ _U(s))$.

Pushforwards, pullbacks and f-maps can easily be extended to presheaves of rings.

We are now in a position to de ne maps between locally ringed spaces, which concludes our formal study of these objects and leads us to one of the most important constructions that we will present.

Definition 56. A *morphism of locally ringed spaces* $(X;O_X)$ and $(Y;O_Y)$ is given by a continuous map $f : X$! Y and an f-map $f^{\#} : O_Y$! O_X such that for all x 2 X the induced map $f_x^{\#} : G_{f(x)}$! F_x is a local ring map, which means that it sends nonunits to nonunits [1, 01HB].

```
structure morphism
{X : Type u} {Y : Type v} [topological_space X] [topological_space Y]
(OX : locally_ringed_space X) (OY : locally_ringed_space Y) :=
(f       : X  !  Y)
(Hf      : continuous f)
(fO      : fmap Hf OX.O.F OY.O.F)
(Hstalks : 8 x s,
    is_unit (fmap.induced OX.O.F OY.O.F fO x s)  !  is_unit s)
```

3.4 Affine schemes

This section formalises the contents of [1, 01HR]. It is here where our work on commutative algebra and our work on sheaf theory meet.

Recall from section 3.1.4 that given a ring R we can de ne a topological space $Spec(R)$. Moreover, we can choose $D(f) = fp$ C R j f 2 pg for each f 2 R which we called standard opens and we showed that $fD(f)g$ forms a standard basis (Lemma 14). Another important property that we proved was that this basis is compact (Lemma 15). We proceeded by looking into induced maps from ring homomorphisms (De nition 17) and put particular emphasis on the induced map : $Spec(R_f)$! $Spec(R)$ which was used to show $Spec(R_f) =$ $D(f)$

The objective of this section is to show that $Spec(R)$ is a locally ringed space and we will need all of this machinery to do so. The rst step is to de ne a presheaf of rings on it, which is often called the *structure presheaf*. Then, we will prove that it is a sheaf applying

Lemma 10 appropriately, which will require some work. Finally, it will follow quite nicely from the de nition that the stalks of the structure presheaf are local rings.

In sections 3.2.4 and 3.2.5 we saw how it is possible to de ne a presheaf on a basis and extend it to the whole space. That is exactly what we will do to de ne a presheaf on $Spec(R)$. In the standard treatment, it is given by the rule $D(f) \mapsto R_f$. We adopt a slightly di erent approach. In Lean we are given some open $U \subseteq Spec(R)$ together with the data `$\exists f \in R; U = D(f)$'. We could obtain this $f$ using `classical.some`. Although in general we use choice freely, it is not required for this speci c de nition so we present a computable de nition of the structure presheaf on $Spec(R)$.

Definition 57. Let R be a ring. The *structure presheaf on the standard basis* $O_{Spec(R)}$ is the presheaf of rings on the basis $B = \{D(f) \mid f \in R\}$ given by the rule $U \mapsto S_U{}^{-1}R$ for $U \in B$ where $S_U = \{r \in R \mid U \subseteq D(r)\}$ [1, 01HR].

```
def S (U : opens (Spec R)) : set R := {r : R | U.1 ⊆ D'(r)}

def structure_presheaf_on_basis
: presheaf_of_rings_on_basis (Spec R) (D_fs_basis R) :=
{ F := λ U BU, localization R (S U),
  res := λ U V BU BV HVU,
    begin
      have H := S.rev_mono HVU,
      apply quotient.lift
      (λ (x : R × (S U)), ⟦(x.1, (⟨x.2.1, H x.2.2⟩ : (S V)))⟧),
      { rintros ⟨a1, b1, Hb1⟩ ⟨a2, b2, Hb2⟩ ⟨t, Ht, Habt⟩; simp,
        use [t, H Ht, Habt] }
    end,
  Hid := λ U BU, funext (λ x, quotient.induction_on x (λ a, by simp)),
  Hcomp := λ U V W BU BV BW HWV HVU,
    funext (λ x, quotient.induction_on x (λ a, by simp)),
  Fring := λ U BU, by apply_instance,
  res_is_ring_hom := λ U V BU BV HVU,
    { map_one := rfl,
      map_add := λ x y,
        quotient.induction_on₂ x y (λ ⟨r₁, s₁, hs₁⟩ ⟨r₂, s₂, hs₂⟩, rfl),
      map_mul := λ x y,
        quotient.induction_on₂ x y (λ ⟨r₁, s₁, hs₁⟩ ⟨r₂, s₂, hs₂⟩, rfl) } }
```

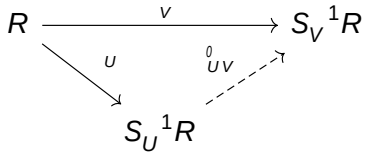
It might not be obvious at rst that this de nition is equivalent to the de nition found in a standard reference. In order to convince ourselves, and Lean, we prove that this de nition satis es the localisation predicate for R away from f. Note that, to prove it, we cannot avoid using the axiom of choice.

Proposition 58. Suppose $U = D(f)$, then $O_{Spec(R)}(U) = R_f$.

```
lemma structure_presheaf.localization
: is_localization_data
    (powers (some BU))
    (of : R → localization R (S U))
```

Proof. Let $\frac{f^n}{1} \in S_U{}^1R$. It suffices to show that $f^n \in S_U$ which translates to $D(f) \subseteq D(f^n)$. For any prime ideal $\mathfrak{p} \subseteq R$ if $f \notin \mathfrak{p}$ then $f^n \notin \mathfrak{p}$, therefore $(L1)$ holds. Now take any $\frac{a}{s} \in S_U{}^1R$, then $D(f) \subseteq D(s)$ so by Lemma 24 there exists $n \in \mathbb{N}$ and $b \in R$ such that $f^n = bs$. Therefore $\frac{f^n}{1}\frac{a}{s} = \frac{ab}{1}$ which establishes $(L2)$. Finally, suppose that $\frac{a}{s} = \frac{0}{1}$, then $at = 0$ for some $t \in S_U$. By the same argument as before we can find $m \in \mathbb{N}$ and $c \in R$ such that $f^m = ct$ so $af^m = 0$ and we have proved $(L3)$. □

Next, we look at the restriction maps $\rho_{UV} : S_U{}^1R \to S_V{}^1R$ where $U = D(f)$, $V = D(g)$ and $V \subseteq U$. There are canonical maps $\rho_U : R \to S_U{}^1R$ and $\rho_V : R \to S_V{}^1R$. Moreover if $s \in S_U$ we have $U \subseteq D(s)$ and since $V \subseteq U$ we deduce that $V \subseteq D(s)$ so $s \in S_V$ or, in other words, ρ_V inverts S_U. By the universal property of localisation we have that the following diagram commutes.



Using the localisation predicate, the map θ_{UV} is given by the following definition.

```
def structure_presheaf_on_basis.res
: localization R (S U) → localization R (S V) :=
is_localization_initial
  (S U)
  (of : R → localization R (S U))
  (of.is_localization_data (S U))
  (of : R → localization R (S V))
  (inverts_data.of_basis_subset BU BV H)
```

It will be useful to prove that this map is definitionally equal to the restriction maps defined on $\mathcal{O}_{Spec(R)}$, that is $\rho_{UV} = \theta_{UV}$.

```
lemma structure_presheaf_on_basis.res_eq
: (structure_presheaf_on_basis R).res =
  @structure_presheaf_on_basis.res R _
```

However, the proof of this lemma follows trivially from the uniqueness condition in the localisation property.

At this point we are in good position to start proving that $O_{Spec(R)}$ is indeed a sheaf on a basis. In general, for any sheaf F we can reformulate the sheaf condition by saying that for all $U \in B$ and all covers $\{U_i\} \subseteq B$ of U the following sequence is exact.

$$0 \to F(U) \to \bigoplus_i F(U_i) \to \bigoplus_{i,j} F(U_i \cap U_j)$$

Here $\to$ is given by the restriction maps $(\rho_{UU_i})_i$ and $\to$ by the differences of the restriction maps on the intersection $\left(\rho_{U_i(U_i \cap U_j)} - \rho_{U_j(U_i \cap U_j)}\right)_{i,j}$. The sequence being exact means that $\to$ is injective, which is precisely the locality condition. Moreover, $\ker(\to) = \mathrm{im}(\to)$ so if sections $s_i \in F(U_i)$ are mapped to zero by $\to$, which means that they coincide in the intersections, then they are in the image of $\to$ so there exists an $s \in F(U)$ such that $s|_{U_i} = s_i$ for all i, in other words, the gluing condition holds.

The plan is to show that $O_{Spec(R)}$ satisfies this condition for any $U = D(f)$ and any open cover $\{U_i\} \subseteq B$ of U, where B is the basis formed by the standard opens. In this case, $O_{Spec(R)}(U) = R_f$ and if $U_i = D(g_i)$ then $O_{Spec(R)}(U_i) = R_{g_i}$. Moreover $U_i \subseteq U$ implies, by Lemma 23, that $\frac{f}{1}$ is invertible in R_{g_i}. This means that it makes sense to think of R_{g_i} as $(R_f)_{\frac{g_i}{1}}$. Similary, $O_{Spec(R)}(U_i \cap U_j) = R_{g_i g_j}$ which we would like to think of as $(R_f)_{\frac{g_i}{1}\frac{g_j}{1}}$. Of course, we will carefully prove these arguments but, assuming they are correct, we are in the following situation:

$$0 \to R_f \to \bigoplus_i (R_f)_{\frac{g_i}{1}} \to \bigoplus_{i,j} (R_f)_{\frac{g_i}{1}\frac{g_j}{1}};$$

which means that we have almost everything we need to apply Lemma 10.

As a matter of fact, we do not even need to worry about the isomorphism $S_U{}^1R = R_f$ which we proved in Lemma 58 as a sanity check. The scenario would then be closer to:

$$0 \to S_U{}^1R \to \bigoplus_i \left(S_U{}^1R\right)_{\frac{g_i}{1}} \to \bigoplus_{i,j} \left(S_U{}^1R\right)_{\frac{g_i}{1}\frac{g_j}{1}};$$

If we look at the input of Lemma 10, we see that in order to make all the arguments above precise we just need to come up with two sets of maps:

- $\rho_i : S_U{}^1R \to S_{U_i}{}^1R$ satifying the localisation predicate for $S_U{}^1R$ away from $\frac{g_i}{1}$, and
- $\rho_{ij} : S_U{}^1R \to S_{U_i \cap U_j}{}^1R$ satifying the localisation predicate for $S_U{}^1R$ away from $\frac{g_i}{1}\frac{g_j}{1}$.

Then $\to$ is explicitly calculated from these. Choose $\rho_i = \rho_{UU_i}$ and $\rho_{ij} = \rho_{U(U_i \cap U_j)}$. We prove that these choices work.

Lemma 59. If $V \subseteq U$ for $V = D(g)$ and $U = D(f)$ then the restricion map of $O_{Spec(R)}$ $\rho_{UV} : S_U{}^1R \to S_V{}^1R$ satisfies the localisation predicate for $S_U{}^1R$ away from $\frac{g}{1}$.

```
lemma structure_presheaf.res.localization
: is_localization_data
    (powers (of (some BV)))
    (structure_presheaf_on_basis.res BU BV H)
```

Proof. Take $\frac{g^n}{1} \in S_U{}^1R$, we need to show that $\left(\frac{g^n}{1}\right)$ is invertible in $S_V{}^1R$. Since $\_$ satisfies the universal property we have that $\left(\frac{g^n}{1}\right) = \frac{g^n}{1}$ so it is invertible by Lemma 58, which shows ($L1$). Now pick any $\frac{a}{s} \in S_V{}^1R$. By Lemma 58 again, $S_V{}^1R = R_g$, so there is $r \in R$ and $n \in \mathbb{N}$ such that $\frac{g^n}{1}\frac{a}{s} = \frac{r}{1}$ in $S_V{}^1R$. Now consider $\frac{g^n}{1}; \frac{r}{1} \in S_U{}^1R$. To prove ($L2$) suffices to show that $\left(\frac{g^n}{1}\right)\frac{a}{s} = \left(\frac{r}{1}\right)$ which, again, follows from the universal property hence ($L2$) is satisfied. Finally, suppose that $\frac{a}{s} \in S_U{}^1R$ and $\left(\frac{a}{s}\right) = \frac{0}{1}$. Working on $S_U{}^1R$, we can multiply through by some power of f such that $\frac{f^n}{1}\frac{a}{s} = \frac{b}{1}$ for some $b \in R$. By the universal property, $\frac{b}{1} = \frac{0}{1}$ in $S_V{}^1R$. Therefore there exists $m \in \mathbb{N}$ such that $g^m b = 0$. By Lemma 24, there exists some $k \in \mathbb{N}$ such that $g^k = cf$ for some $c \in R$. We claim that $\frac{g^{kn+m}}{1}\frac{a}{s} = \frac{0}{1}$, which is enough to establish ($L3$). But this is obvious becasue $\frac{g^{kn+m}}{1}\frac{a}{s} = \frac{c^n f^n}{1}\frac{g^m}{1}\frac{a}{s} = \frac{c^n}{1}\frac{g^m b}{1} = \frac{0}{1}$. □

For clarification, we invoke the universal property on the canonical map $\__U : R \to S_U{}^1R$ sending $a \mapsto \frac{a}{1}$ and $\__{UV}$. Since, as we showed, $\__U$ satisfies the localisation predicate, we have $\__{UV} \circ \__U = \__V$. The last proof and the one that follows are fairly technical and somewhat trivial. Textbooks never go into so much detail and tend to claim that the sheaf condition follows immediately from the gluing lemma. Nevertheless, we believe it is important to show the inner workings of our approach and see the localisation predicate in action.

Lemma 60. If $V \setminus W \subseteq U$ for $W = D(h)$, $V = D(g)$ and $U = D(f)$ then the restricion map $\__{U(V \setminus W)} : S_U{}^1R \to S_{V \setminus W}{}^1R$ of $\mathcal{O}_{Spec(R)}$ satisfies the localisation predicate for $S_U{}^1R$ away from $\frac{g}{1}\frac{h}{1}$.

```
def structure_presheaf_on_basis.res_to_inter
: localization R (S U) → localization R (S (V \ W)) :=
structure_presheaf_on_basis.res BU (BVW BV BW) (HVWU HVU)

lemma structure_presheaf.res_to_inter.localization
: is_localization_data
    (powers ((of (some BV)) * (of (some BW))))
    (structure_presheaf_on_basis.res_to_inter BU BV BW HVU)
```

Before we start the proof, it is worth explaining one of its main complications. We know that $V \setminus W \in \mathcal{B}$ so we can use choice to deduce that $V \setminus W = D(r)$ for some $r \in R$ and apply, as we did before, the fact that $S_{V \setminus W}{}^1R = R_r$. It does not seem very useful at first because r appeared from nowhere. However, Lemma 12 (3) tells us that $V \setminus W = D(gh)$. One needs to be a bit careful here because from $D(r) = D(gh)$ we cannot deduce that $r = gh$. Instead, we will need to work with Lemma 24 in both directions. This result gives us that $(gh)^n = ar$ for some $a \in R$ and $n \in \mathbb{N}$, and $r^m = bgh$ for some $b \in R$ and $m \in \mathbb{N}$.

Proof of 60. Take $\frac{g^k}{1}\frac{h^k}{1} \in S_U{}^1R$. Then, multiplying though by $\frac{b^k}{1}$ we consider $\frac{r^{km}}{1}$ and, by Lemma 59 we can find $\frac{w}{s} \in S_{V \setminus W}{}^1R$ such that $\left(\frac{r^{km}}{1}\right)\frac{w}{t} = \frac{1}{1}$. Therefore, since the restriction map is a ring homomorphism, $\left(\frac{b^k}{1}\right)\frac{w}{t}$ is an inverse for $\left(\frac{g^k}{1}\frac{h^k}{1}\right)$, which proves ($L1$). Proceed by choosing any $\frac{x}{s} \in S_{V \setminus W}{}^1R$. Again, by Lemma 59, we can find k and $y \in R$

such that $\left(\frac{r^k}{1}\right)\frac{x}{s} = \left(\frac{y}{1}\right)$. Multiply through by $\left(\frac{a^k}{1}\right)$ to obtain $\left(\frac{g^k}{1}\frac{h^k}{1}\right)\frac{x}{s} = \left(\frac{a^k y}{1}\right)$. We have thus proved $(L2)$. To show $(L3)$, pick $\frac{x}{s} \in S_U{}^1R$ and suppose that $\left(\frac{x}{s}\right) = \frac{0}{1}$. One more time, we use Lemma 59 and obtain $\frac{r^k}{1}\frac{x}{s} = \frac{0}{1}$. Multiplying through by $\frac{a^k}{1}$ gives $\frac{g^k}{1}\frac{h^k}{1}\frac{x}{s} = \frac{0}{1}$. □

Finally, we have the main pieces to prove the sheaf condition.

Theorem 61. For any ring R, the structure presheaf $\mathcal{O}_{Spec(R)}$ is a sheaf on the standard basis $B = \{D(f) \mid f \in R\}$.

```
theorem structure_presheaf_on_basis_is_sheaf_on_basis
: is_sheaf_on_standard_basis
    (D_fs_standard_basis R)
    (structure_presheaf_on_basis R).to_presheaf_on_basis
```

Proof. By Lemma 47 it is enough to consider finite subcovers to prove the sheaf condition if covers of basis elements can be refined with finite subcovers, which we proved in Lemma 22. Hence, pick any $U \in B$ and any finite open cover $\{U_1, \ldots, U_n\}$ of U. Suppose $U = D(f)$ and $U_i = D(g_i)$. The localisation map $R \to R_f$ induces homeomorphism $D(f) = Spec(R_f)$. We first check that $Spec(R_f) = D\left(\frac{g_1}{1}\right) \cup \cdots \cup D\left(\frac{g_n}{1}\right)$. Pick $\mathfrak{p} \in Spec(R_f)$. Clearly, we have $(\mathfrak{p}) \in D(f) = D(g_1) \cup \cdots \cup D(g_n)$ so $(\mathfrak{p}) \in D(g_i)$ for some i. Therefore, $\mathfrak{p} \in {}^1(D(g_i)) = D\left(\frac{g_i}{1}\right)$ where the last equality follows from the observation made in Lemma 18. We now show that $\frac{1}{1} \in \left(\frac{g_1}{1}, \ldots, \frac{g_n}{1}\right) \subseteq R_f$. By the basic properties of $Spec$ stated in Lemma 12, this is true if and only if $V\left(\frac{g_1}{1}, \ldots, \frac{g_n}{1}\right) = \emptyset$ but $Spec(R_f) \cap V\left(\frac{g_1}{1}, \ldots, \frac{g_n}{1}\right) = D\left(\frac{g_1}{1}\right) \cup \cdots \cup D\left(\frac{g_n}{1}\right) = Spec(R_f)$.

The result we just proved together with the localisation predicates for the restriction maps proved in Lemma 59 and Lemma 60 are exactly the arguments that Lemma 10 takes. Therefore, by our previous discussion, $\mathcal{O}_{Spec(R)}$ is a sheaf on the standard basis B. □

Recall from section 3.3 that defining a sheaf of rings on a topological space is not enough to have a locally ringed space. We are missing the condition that the stalks of $\mathcal{O}_{Spec(R)}$ are local rings.

For any $\mathfrak{p} \in Spec(R)$, there is an map $R \to \mathcal{O}_{Spec(R),\mathfrak{p}}$ given by $x \mapsto \left(Spec(R), \frac{x}{1}\right)$.

```
def to_structure_presheaf_on_basis_stalk : R →
stalk_of_rings_on_standard_basis Bstd (structure_presheaf_on_basis R) P
:=  λ x,
⟨{ U  := opens.univ,
   BU := (D_fs_standard_basis R).1,
   Hx := set.mem_univ P,
   s  := (of : R → localization R (S (opens.univ))) x }⟩
```

The fact that $\mathcal{O}_{Spec(R),\mathfrak{p}}$ is a local ring will be a consequence of it satisfying a specific localisation predicate.

Lemma 62. With the map de ned above, for any prime p C R, the stalk of the structure presheaf on the standard basis $\mathcal{O}_{Spec(R);p}$ satis es the localisation predicate for R at p.

```
lemma structure_presheaf_on_basis_stalk.localization
: is_localization_data
    (-P.1 : set R)
    (to_structure_presheaf_on_basis_stalk P)
```

Proof. We follow the usual pattern and start by proving (L1). Pick s 2 p. We need to nd an inverse for $(Spec(R); \frac{s}{1})$. We have that p 2 $D(s)$. Moreover, it is obvious that s 2 $S_{D(s)}$ so it is invertible in $\mathcal{O}_{Spec(R)}(D(s)) = S_{D(s)}^{\ 1}R$. The desired inverse is therefore $(D(s); \frac{1}{s})$. Given any $(U; \frac{s}{1})$ 2 $\mathcal{O}_{Spec(R);p}$ it follows that $\frac{s}{1}$ 2 $S_U{}^1R$. Thus there is a 2 R and t 2 S_U such that $\frac{t}{1}\frac{s}{1} = \frac{a}{1}$. In particular, t 2 S_U means that U $D(t)$ so p 2 $D(t)$ and, thus, t 2 p so (L2) is proved. Suppose $(U; \frac{s}{1}) = (Spec(R); \frac{0}{1})$. There exists V U with p 2 V where $\frac{s}{1}\big|_V = \frac{0}{1}$. The universal property of localisation allows us to write $\frac{s}{1} = \frac{0}{1}$. Hence there exists t 2 S_V such that $ts = 0$. By the same argument as before, t 2 p and (L3) follows. □

Let us try to understand this localisation predicate. In section 3.2.3 we showed that stalks satisfy a certain universal property. Transporting it to this speci c case, what we said is that for any f 2 p there is a map R_f ! $\mathcal{O}_{Spec(R);p}$ such that all the diagrams commute and such that for any other ring S with this property there exists a unique homomorphism $\mathcal{O}_{Spec(R);p}$! S. We also explained how this construction could be understood as gluing. We would expect the ring that glues together all the R_f's such that f 2 p to be a ring in which every f 2 p is invertible and nothing else, that is precisely R_p.

When we brie y discussed local rings in section 3.1.2 we proved that any ring satisfying the localisation predicate for R at p was a local ring. So a direct application of Lemma 9 gives us the result that we need.

```
lemma structure_presheaf_on_basis_stalk.local
: is_local_ring (to_structure_presheaf_on_basis_stalk P)
```

Finally, we extend $\mathcal{O}_{Spec(R)}$ to the whole space, using the de nition introduced in section 3.2.5. Hereafter, $\mathcal{O}_{Spec(R)}$ denotes the ring on the whole space rather than just de ned on the basis.

```
def structure_sheaf.presheaf (R : Type u) [comm_ring R] :=
presheaf_of_rings_extension
  (D_fs_standard_basis R)
  (structure_presheaf_on_basis R)
```

It follows from Lemma 49 that the extension is a sheaf of rings.

```
theorem structure_presheaf_is_sheaf_of_rings (R : Type u) [comm_ring R]
: is_sheaf_of_rings (structure_sheaf.presheaf R)
```

Therefore, we can give a proper de nition of the *structure sheaf* on $Spec(R)$.

```
def structure_sheaf (R : Type u) [comm_ring R]
: sheaf_of_rings (Spec R) :=
{ F        := structure_sheaf.presheaf R,
  locality := (structure_presheaf_is_sheaf_of_rings R).1,
  gluing   := (structure_presheaf_is_sheaf_of_rings R).2 }
```

We showed in section 3.2.5 that this extension respects the basis elements hence we still have that $O_{Spec(R)}(D(f)) = R_f$. More importantly, by Lemma 50, the extension also respects stalks so $O_{Spec(R);\mathfrak{p}} = R_{\mathfrak{p}}$ for all $\mathfrak{p} \subset R$. The following lemma is an immediate consequence of that and Proposition 5.

```
lemma structure_sheaf.stalk_local
: is_local_ring (stalk_of_rings (structure_sheaf R).F P)
```

We conclude this section with the result we have been building towards which is, without doubt, the biggest achievement of this project.

Definition 63. For any ring R, $(Spec(R); O_{Spec(R)})$ has the structure of a locally ringed space [1, 01HU].

```
def Spec.locally_ringed_space : locally_ringed_space (Spec R) :=
{ O       := structure_sheaf R,
  Hstalks :=   P, structure_sheaf.stalk_local P }
```

3.5 The type `scheme`

In this nal section we give our formal de nition of a scheme.

Definition 64. A *scheme* is a locally ringed space $(X; O_X)$ with an open cover $\{U_i\}$ such that the locally ringed space $(U_i; O_X|_{U_i})$ is isomorphic to $(Spec(R); O_{Spec(R)})$ for some ring R [1, 01IJ].

```
structure scheme (X : Type u) [topological_space X] :=
(carrier    : locally_ringed_space X)
(Haffinecov : ∃ (OC : covering.univ X),
  ∀ i, ∃ (R : Type v) [comm_ring R]
         (fpU : open_immersion_pullback (Spec R) carrier.O.F),
     fpU.range = OC.Uis i
   ^ fpU.carrier = structure_sheaf.presheaf R)
```

Note that we use open pullbacks to express the restriction of the sheaf on the whole space to the sheaf an open subset as it was hinted in section 3.3.

The first scheme that we can trivially define is the empty scheme.

```
def empty_scheme : scheme empty
```

Nonetheless, our main examples of a schemes are *affine schemes*, which are locally ringed spaces isomorphic to $(Spec(R); O_{Spec(R)})$ for some R [1, 01HW].

We conclude our exposition defining the scheme $(Spec(R); O_{Spec(R)})$.

```
variables (R : Type u) [comm_ring R]

def affine_scheme : scheme (Spec R)
```

It is a locally ringed space by Definition 63. Now, we simply choose the cover $\{Spec(R)\}$ and create an open immersion pullback from the identity map.

Chapter 4

Future Work

In this chapter we explain the limitations of the project by discussing two possible next steps. The first has recently been formalised by Kenny Lau and his work revealed some holes the existing API, mainly some lemmas about locally ringed spaces were missing. The second is slightly more ambitious and we suggest a possible roadmap. The treatment of this part is more informal and some knowledge of category theory and slightly more advanced algebra is assumed.

4.1 Spec-Γ adjointness

As it was mentioned in section 1.2, there is a correspondance between the category of affine schemes and the category of commutative rings [1, 01I1]. It turns out that *Spec* is a contravariant functor $\mathbf{CRing} \to \mathbf{Sch_{Aff}}$, where $\mathbf{CRing}$ denotes the category of commutative rings and $\mathbf{Sch_{Aff}}$ the category of affine schemes. We have not developed this terminology in this project but what it means is that for each ring R we obtain an affine scheme $(Spec(R); O_{Spec(R)})$ and, for every ring homomorphism $f : A \to B$, a morphism of affine schemes $Spec(f) : (Spec(B); O_{Spec(B)}) \to (Spec(A); O_{Spec(A)})$ that respects the identity and composition. The map is the one given in Definition 17. Then some work needs to be done to prove that it can be used to define an f-map which satisfies the conditions of Definition 56.

Recall that $(U; F)$ is the same as $F(U)$. In particular, we can consider the global sections $(Spec(R); O_{Spec(R)})$. We can also show that Γ is a contravariant functor $\mathbf{Sch_{Aff}} \to \mathbf{CRing}$. The claim now is that *Spec* and Γ are adjoint functors. This means that they induce bijections between the sets of homomorphisms in the two categories. Formally, if A is a commutative ring and $(Spec(B); O_{Spec(B)})$ an affine scheme then we have

$$\mathrm{Hom}_{\mathbf{CRing}}(A; \Gamma(Spec(B); O_{Spec(B)})) = \mathrm{Hom}_{\mathbf{Sch_{Aff}}}((Spec(B); O_{Spec(B)}); (Spec(A); O_{Spec(A)})).$$

Recall that in the definition of the structure sheaf on $Spec(R)$ (Definition 57) the core idea was that $D(f) \stackrel{?}{=} R_f$ although, strictly speaking, our definition was isomorphic, but not equal, to R_f. Hence $\Gamma(Spec(R); O) = R$ so we can rewrite the statement above as

$$\mathrm{Hom}_{\mathbf{CRing}}(A; B) = \mathrm{Hom}_{\mathbf{Sch_{Aff}}}((Spec(B); O_{Spec(B)}); (Spec(A); O_{Spec(A)})).$$

This is the result that we expected. It says that there is a one-to-one arrow-reversing correspondence between maps of rings and maps of a ne schemes. In fact, we can say even more. Since every object in $\mathbf{Sch}_{\mathbf{Aff}}$ is isomorphic to $(Spec(R); O_{Spec(R)})$ for some R we say that *Spec* is an essentially surjective functor which implies that $\mathbf{CRing}$ and $\mathbf{Sch}_{\mathbf{Aff}}$ are equivalent categories.

In [1, 01I2] this is deduced from adjointness in a more general setting. We can work in $\mathbf{LRS}$, the category of locally ringed spaces, and prove that for $(X; O_X)$ a locally ringed space

$$\mathrm{Hom}_{\mathbf{CRing}}(A; \ (X; O_X)) = \mathrm{Hom}_{\mathbf{LRS}}((X; O_X); (Spec(A); O_{Spec(A)})):$$

As a matter of fact, we can replace $(Spec(A); O_{Spec(A)})$ by any a ne scheme $(Y; O_Y)$ and the statement becomes

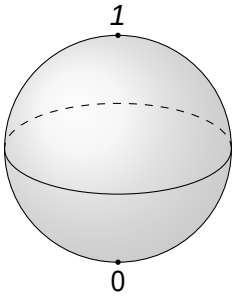
$$\mathrm{Hom}_{\mathbf{CRing}}(\ (Y; O_Y); \ (X; O_X)) = \mathrm{Hom}_{\mathbf{LRS}}((X; O_X); (Y; O_Y)):$$

Lau has recently proved this statement and we plan on incorporating it into the existing codebase. However, it raised some concerns regarding the API for locally ringed spaces. Because our main goal was to give a correct de nition of a scheme, when we were at the stage of de ning a locally ringed space we gave the de nition of the objects and the morphisms. However, thanks to this test, we realised that some basic properties about them should be included in order to make them usable. Therefore, the rst next step is to make sure that a comprehensive API for locally ringed spaces is provided.

4.2 Projective schemes

The rst question that anyone in the eld of formal veri cation asks when you tell them that you have de ned a complicated type is whether you are able to give examples of objects of that type. In our case, we showed that a ne schemes are schemes. However, it uses the cover condition in a trivial way that does not capture its power. For this reason, we think it is important to give an example of a non-a ne scheme [1, 01ND].

Let us consider how one could show that the projective line $\mathbf{P}^1(\mathbb{C})$ has the structure of a scheme. But, what is $\mathbf{P}^1(\mathbb{C})$? Is it a complex manifold, a sphere in $\mathbb{R}^3$ or a scheme? It seems rather convenient to say that it is a scheme but, in that case, how do we know that we have de ned $\mathbf{P}^1(\mathbb{C})$ and not something else? The rst issue is that the de nition requires a topological space as input. A possibility is to choose $X = (Spec(\mathbb{C}[x]) \ t \ Spec(\mathbb{C}[y]))=$ where the relation identi es $D(x)$ $Spec(\mathbb{C}[x])$ with $D(y)$ $Spec(\mathbb{C}[y])$. By a previous result, $D(x) = Spec\left(\mathbb{C}\left[x; \frac{1}{x}\right]\right)$ and $D(y) = Spec\left(\mathbb{C}\left[y; \frac{1}{y}\right]\right)$. The identi cation is given by the $\mathbb{C}$-algebra isomorphism $y \mapsto \frac{1}{x}$. It would require some work but we could prove that X is a perfectly valid topological space as it is a quotient of a disjoint union of spaces.



The idea is that *Spec*(C[*x*]) = *f*(*x*) *j* *2* C*g* [*f*(0)*g* so it looks like C with an extra point, whose closure happens to be the whole space and is called a *generic point*. If we disregard it, we can think of *Spec*(C[*x*]) as the a ne line and, in fact, that is how $\mathbf{A}^1$(C) is de ned in scheme theory. We now take two copies of the a ne line and glue them in a way such that the `zero' of the second copy, which we will call *1* *2* *X* and is represented by (*y*) in the construction, behaves like in nity. The gluing function expresses exactly that by assigning points near *1* in the second copy to large values in the rst copy.

Then we can augment the two components of *X* with the structure of an a ne scheme as we did in section 3.4. Now the challenge is to glue together the two sheaves $\mathcal{O}_{Spec(\mathbb{C}[x])}$ and $\mathcal{O}_{Spec(\mathbb{C}[y])}$ to get a sheaf $\mathcal{O}_X$ and prove that $(X; \mathcal{O}_X)$ is a locally ringed space. If we do so we will have proved that *X* has the structure of a scheme and so we will have given a de nition for $\mathbf{P}^1$(C).

The suggested test is to prove that this scheme is not isomorphic to any a ne scheme as it is done in [1, 01JE]. By doing so we will not be able to say with absolute certainty that our de nition corresponds to the mathematical notion of the projective line but we will have shown that we can de ne a non-a ne scheme. They use the relationship between *Spec* and discussed earlier. If $(X; \mathcal{O}_X)$ was an a ne scheme then *Spec*($(X; \mathcal{O}_X)$) = *X*. However, it turns out that $(X; \mathcal{O}_X)$ = C because the global sections need to respect the gluing function so they are the polynomials *f*(*x*) *2* C[*x*] such that $f\left(\frac{1}{y}\right)$ is a polynomial in C[*y*] and are therefore the constant polynomials. Hence *Spec*($(X; \mathcal{O}_X)$) consists of one point whereas *X* is in nite.

In order to successfully achieve this, one needs to build a robust infrastructure to glue sheaves. This is currently being done and we are experimenting with several approaches as it is not clear how to conveniently handle sheaf restrictions. A detailed explanation of the mathematical details is given in [1, 00AK].

Alternatively, we could develop projective schemes by de ning *Proj*, the projective counterpart of *Spec*. We would need to assume that *S* is a graded ring and de ne *Proj*(*S*) to be the set of homogeneous prime ideals. There is an analogous construction to the one given for a ne schemes explained in [1, 01M3]. After developing all of this machinery, we could de ne the projective line $\mathbf{P}^1$(C) to be the scheme $\left(Proj(\mathbb{C}[x]); \mathcal{O}_{Proj(\mathbb{C}[x])}\right)$. We believe that this is a more long term plan that will require much more e ort as, for instance, graded rings are not in mathlib yet. In addition, since for now the objective is, essentially, to give an example of a non-a ne scheme, the test described above is enough to achieve it.

Chapter 5

Conclusion

This project lives in one of the many areas of intersection between mathematics and computer science. Along the way, we have discovered that the two communities are making a tremendous effort to understand each other's points of view. On the one hand, experts in formal verification are finding mechanisms to make theorem provers operate closer to the way a mathematician would. On the other hand, despite the rather steep learning curve of these systems, the mathematics community is getting more and more involved and realising that formalisation will play an important role in the next few decades. However, there is still some scepticism regarding the viability of formalising mathematics. The only way to overcome these objections is to keep improving proof assistants both in terms of capabilities and usability and use them to prove important results such as the one described in section 1.1.

Giving a formal definition of a scheme in Lean is a step in that direction. It serves as an example that theorem provers can handle modern mathematics. Furthermore, it shows that it is possible to learn about algebraic geometry and theorem proving at the same time and how this way of learning new concepts does not necessarily slow down the learning process but can, instead, provide a deeper understanding.

Bibliography

[1] The Stacks Project Authors. *The Stacks Project*;. https://stacks.math.columbia.edu (Accessed: 19/06/2019).

[2] Harrison J, Urban J, Wiedijk F. *History of Interactive Theorem Proving*. Handbook of the History of Logic. 2014 12;9:135{214. Available from: https://doi.org/10.1016/B978-0-444-51624-4.50004-6.

[3] Geuvers H. *Proof assistants: History, ideas and future*. Sadhana. 2009 Feb;34(1):3{25. Available from: https://doi.org/10.1007/s12046-009-0001-5.

[4] Hales TC. *The Kepler conjecture*. arXiv Mathematics e-prints. 1998 Nov;Available from: https://arxiv.org/abs/math/9811078v1.

[5] Hales TC. *A Proof of the Kepler Conjecture*. Annals of Mathematics. 2005;162(3):1065{1185. Available from: http://www.jstor.org/stable/20159940.

[6] Hales T, Adams M, Bauer G, Dang TD, Harrison J, Le Truong H, et al.; Cambridge University Press. *A formal proof of the Kepler conjecture*. Forum of Mathematics, Pi. 2017;5. Available from: https://doi.org/10.1017/fmp.2017.1.

[7] Hales TC. *Developments in Formal Proofs*. arXiv e-prints. 2014 Aug;Available from: https://arxiv.org/abs/math/1408.6474.

[8] Paulson LC. *A machine-assisted proof of Gödel's incompleteness theorems for the theory of hereditarily nite sets*. The Review of Symbolic Logic. 2014;7(3):484{498. Available from: https://doi.org/10.1017/S1755020314000112.

[9] Gonthier G. *Formal proof{the four-color theorem*. Notices of the AMS. 2008;55(11):1382{1393.

[10] Gonthier G, Asperti A, Avigad J, Bertot Y, Cohen C, Garillot F, et al. *A Machine-Checked Proof of the Odd Order Theorem*. In: Blazy S, Paulin-Mohring C, Pichardie D, editors. Interactive Theorem Proving. Berlin, Heidelberg: Springer Berlin Heidelberg; 2013. p. 163{179.

[11] Lewis RY. *A formal proof of Hensel's lemma over the p-adic integers*. In: Proceedings of the 8th ACM SIGPLAN International Conference on Certi ed Programs and Proofs. ACM; 2019. p. 15{26.

[12] Dahmen SR, Hölzl J, Lewis RY. *Formalizing the Solution to the Cap Set Problem (preprint)*. 2019;Available from: https://lean-forward.github.io/e-g/e-g.pdf.

[13] *The QED Manifesto*. In: Proceedings of the 12th International Conference on Automated Deduction. CADE-12. London, UK, UK: Springer-Verlag; 1994. p. 238{251. Available from: http://dl.acm.org/citation.cfm?id=648231.752823.

[14] Dieudonne J. *The Historical Development of Algebraic Geometry*. The American Mathematical Monthly. 1972;79(8):827{866. Available from: http://www.jstor.org/stable/2317664.

[15] Grothendieck A. *Elements de geometrie algebrique: I. Le langage des schemas*. Publications Mathematiques de l'IHES. 1960;4:5{228.

[16] Gannon T. *What is a scheme?* Notices of the AMS. 2017;64(11):1300{1301.

[17] Avigad J, de Moura L, Kong S. *Theorem proving in Lean*. Microsoft Research; 2017. https://leanprover.github.io/tutorial/tutorial.pdf (Accessed: 19/06/2019).

[18] Avigad J, Ebner G, Ullrich S. *The Lean Reference Manual*. Microsoft Research; 2018. https://leanprover.github.io/reference/lean_reference.pdf (Accessed: 19/06/2019).

[19] de Moura L, Kong S, Avigad J, Van Doorn F, von Raumer J. *The Lean theorem prover (system description)*. In: International Conference on Automated Deduction. Springer; 2015. p. 378{388.

[20] Lewis RY. *Two Tools for Formalizing Mathematical Proofs*. 2018;https://robertylewis.com/files/dissertation.pdf (Accessed: 19/06/2019).

[21] Martin-Lof P, Samnni G. *Intuitionistic type theory*. Napoli: Bibliopolis; 1984.

[22] Nederpelt R, Geuvers H. *Type theory and formal proof: an introduction*. Cambridge University Press; 2014.

[23] Hartshorne R. *Algebraic geometry*. vol. 52. Springer Science & Business Media; 2013.

[24] Atiyah M. *Introduction to commutative algebra*. CRC Press; 2018.